OpenZFS New Features Including Direct I/O and Compression/Erasure Offloads

Brian Atkinson, Jason Lee - LANL

Kelly Ursenbach - Eideticom

May 22, 2023

LA-UR-23-25110

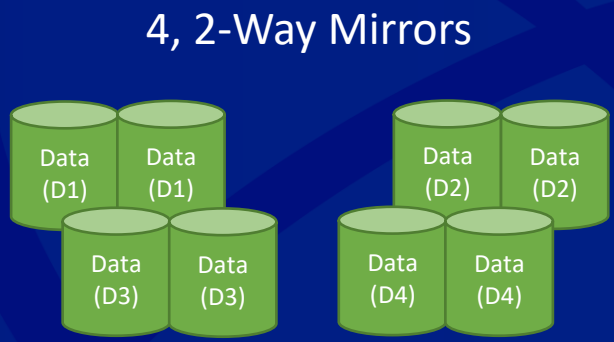
Los Alamos NATIONAL LABORATORY EIDETICOM

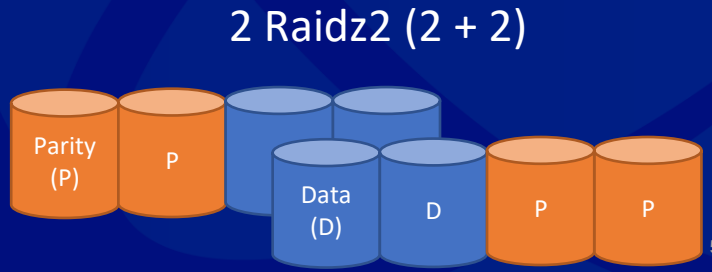
Agenda

- Discuss OpenZFS developments to improve performance with NVMe Devices
- There are two parts to this BoF
  - Part 1: Addition of O_DIRECT to OpenZFS
  - Part 2: Computational Storage Offloads to OpenZFS

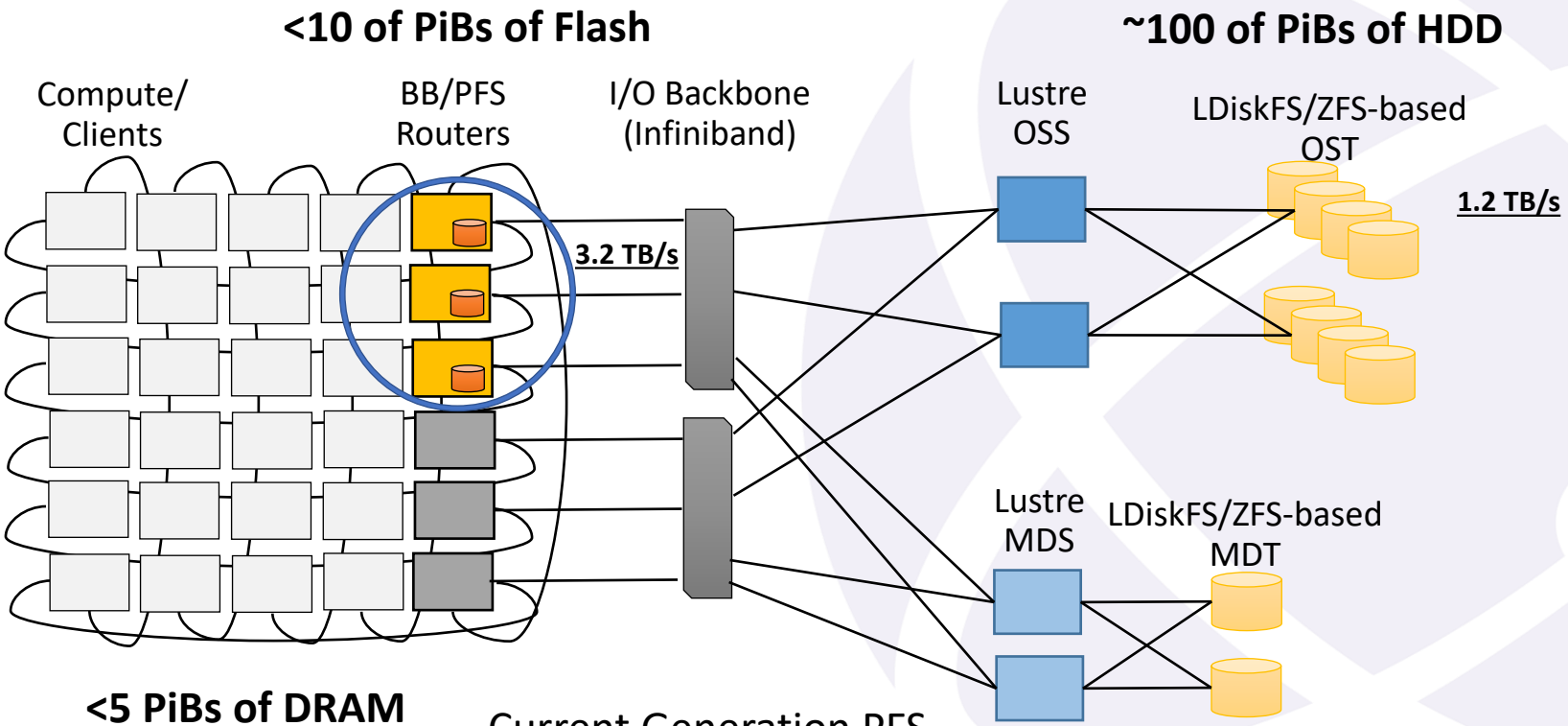
Why Does LANL Care about ZFS?

- **One of two available backing FS's for Lustre**
- Open sourced
- High integrity (Flag in the ground)
  - Erasure coding (raidz)
  - Mirrors
  - Checksums
  - Snapshots
- Feature rich
  - Compression
  - Dedup
  - Encryption
- ZFS traditionally has good performance with HDD

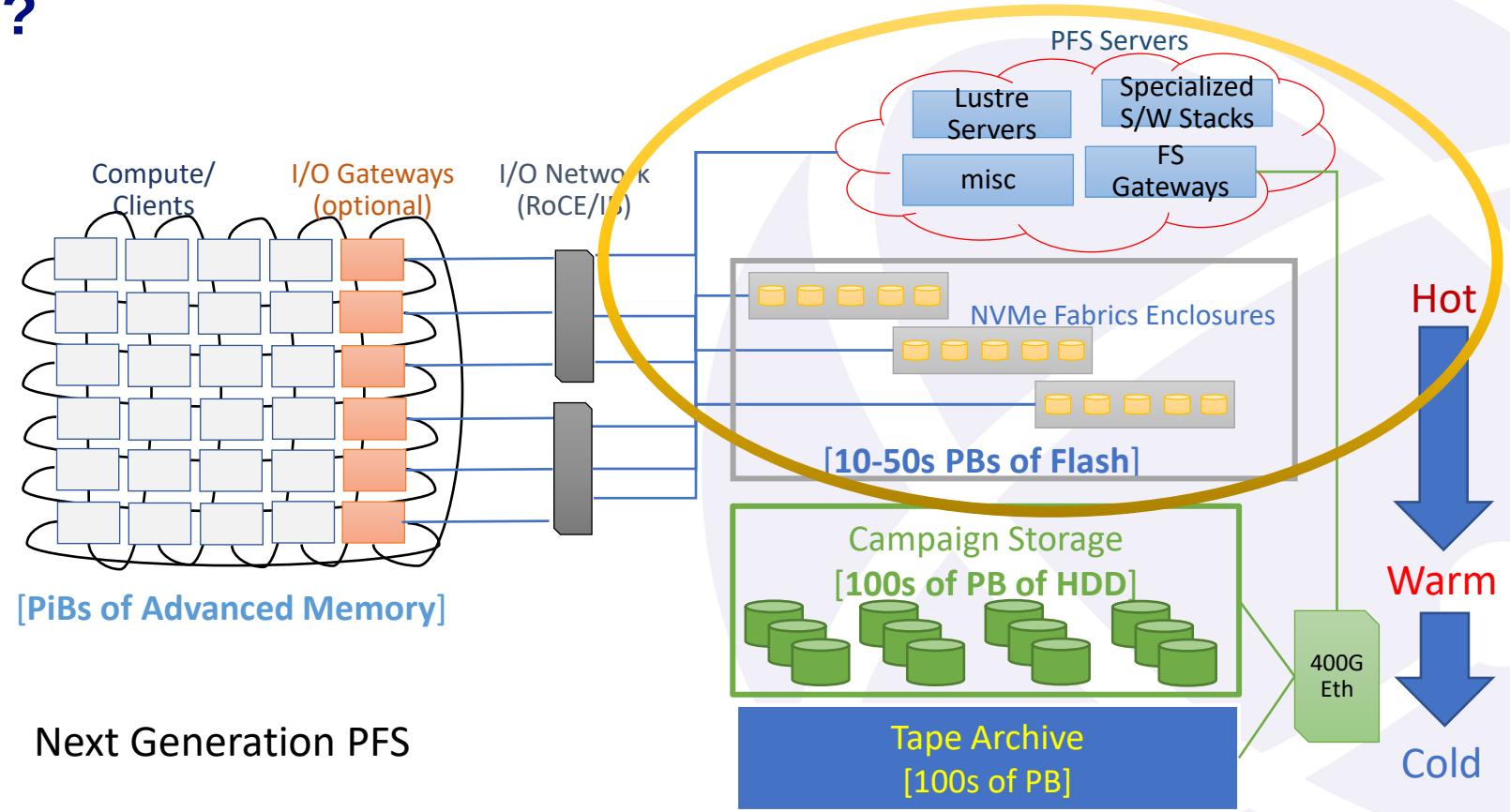




Why are we focused on NVMe Device Performance with ZFS?

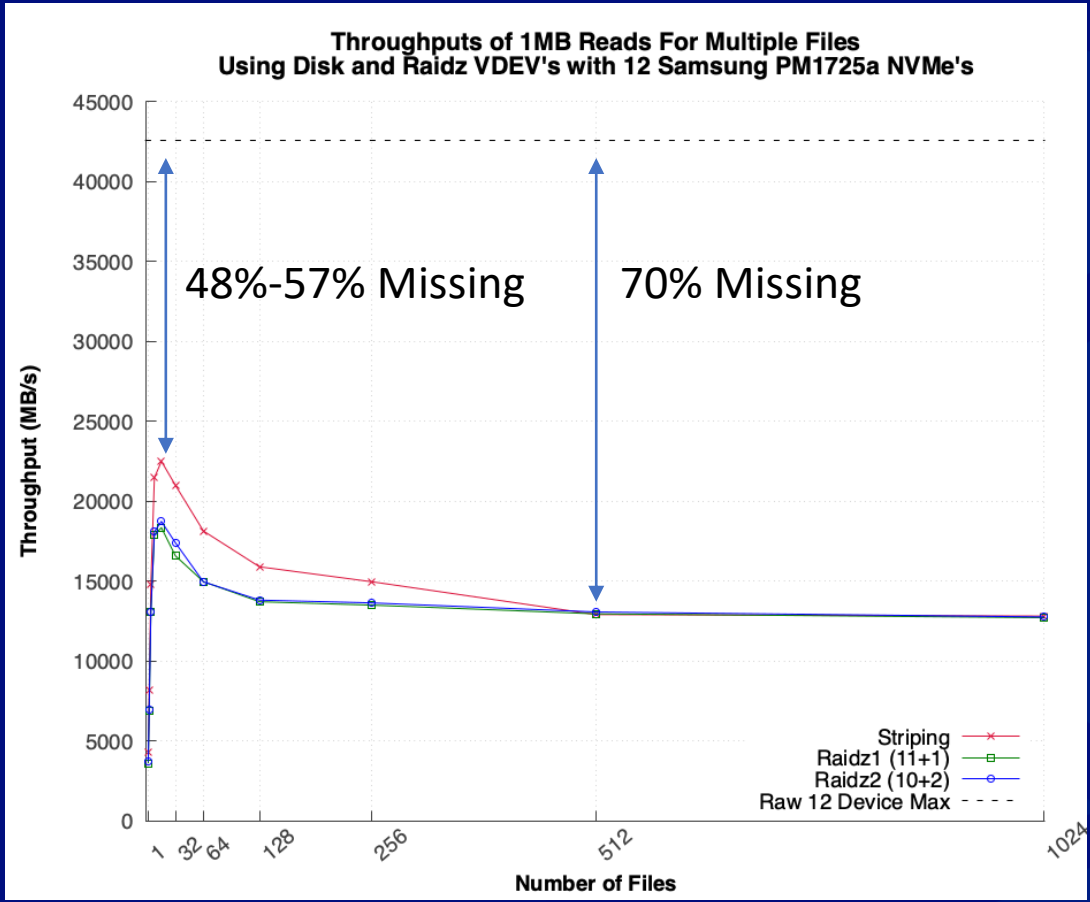


Why are we focused on NVMe Device Performance with ZFS?

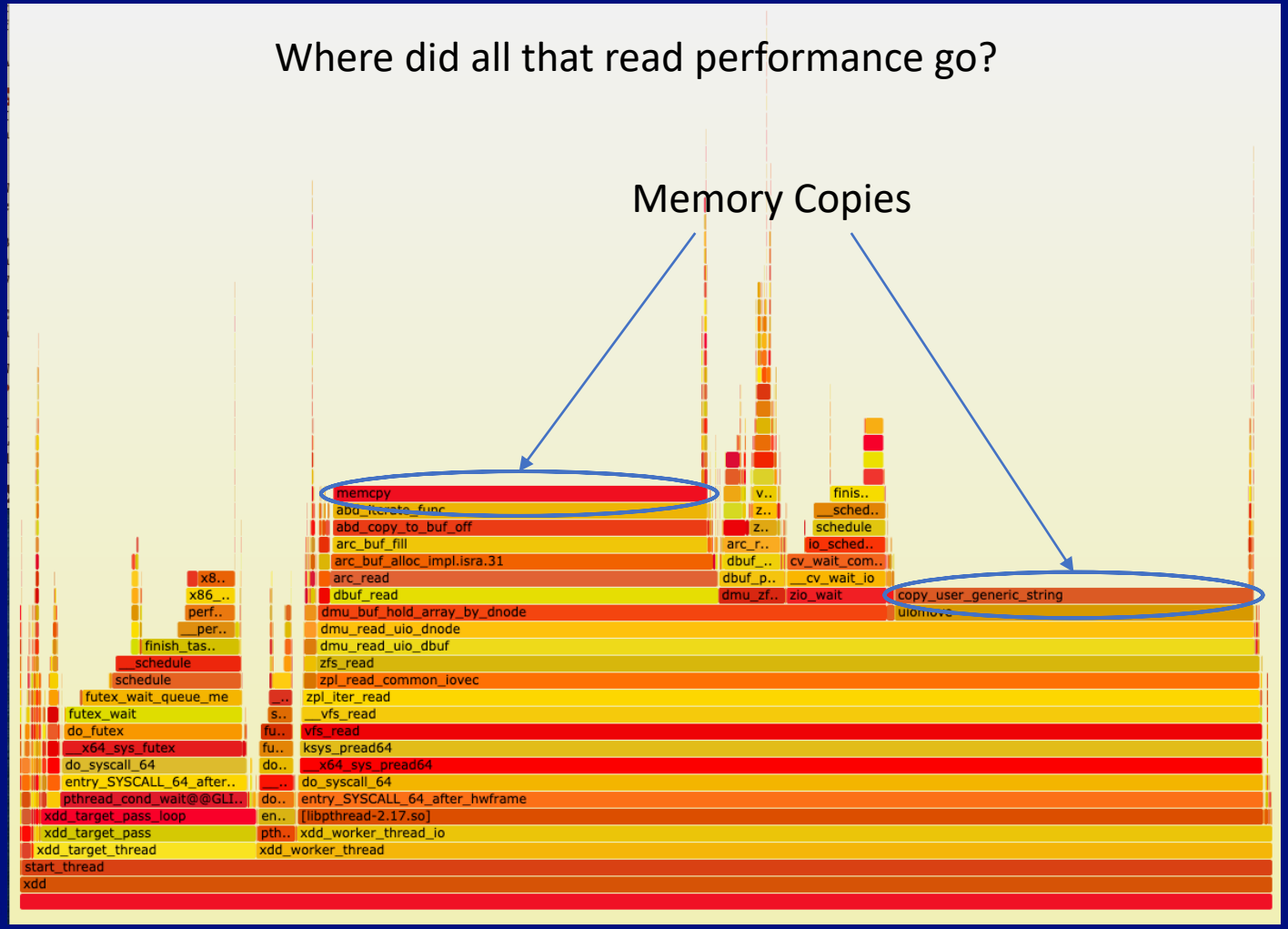


Next Generation PFS

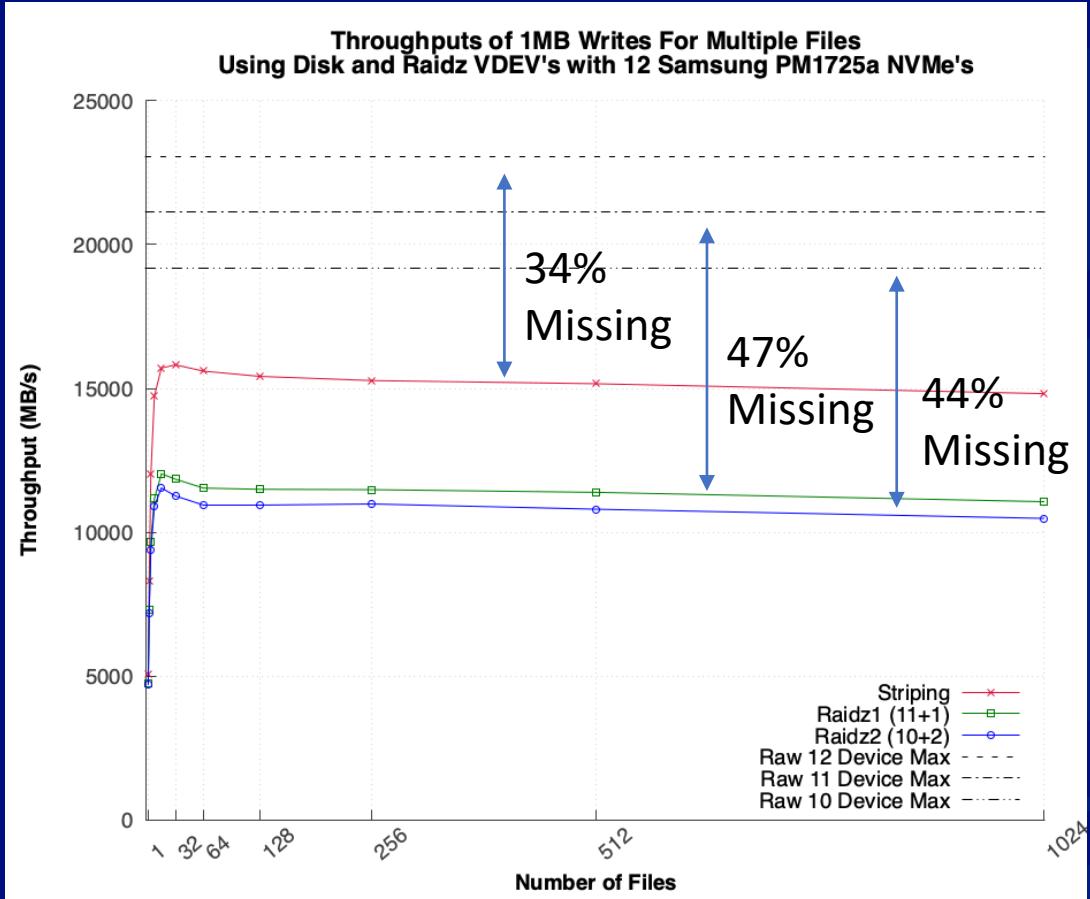
ZFS gen3 NVMe Zpools Sequential Read Performance



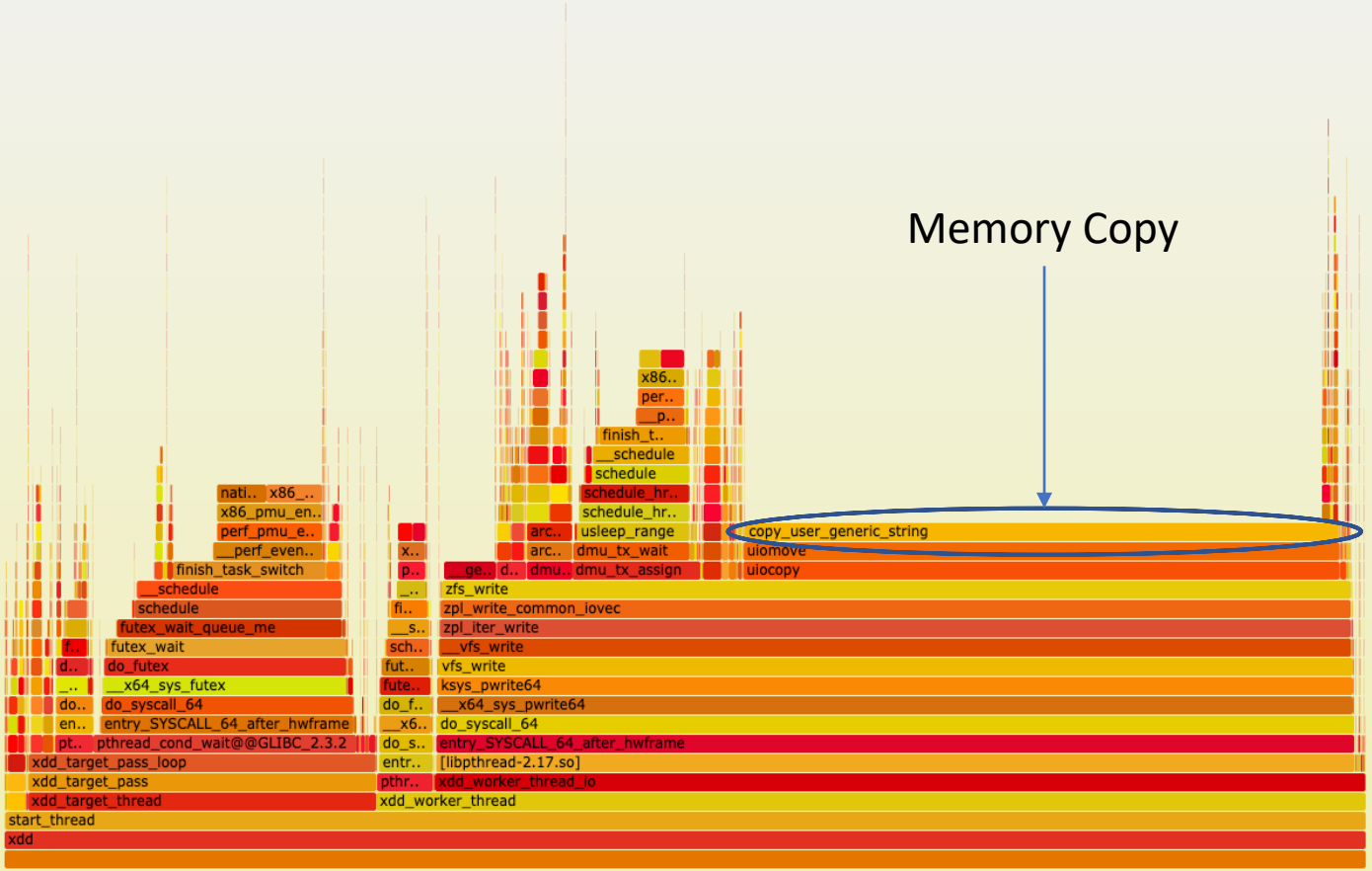
Where did all that read performance go?



ZFS gen3 NVMe Zpools Sequential Write Performance



Where did all that write performance go?



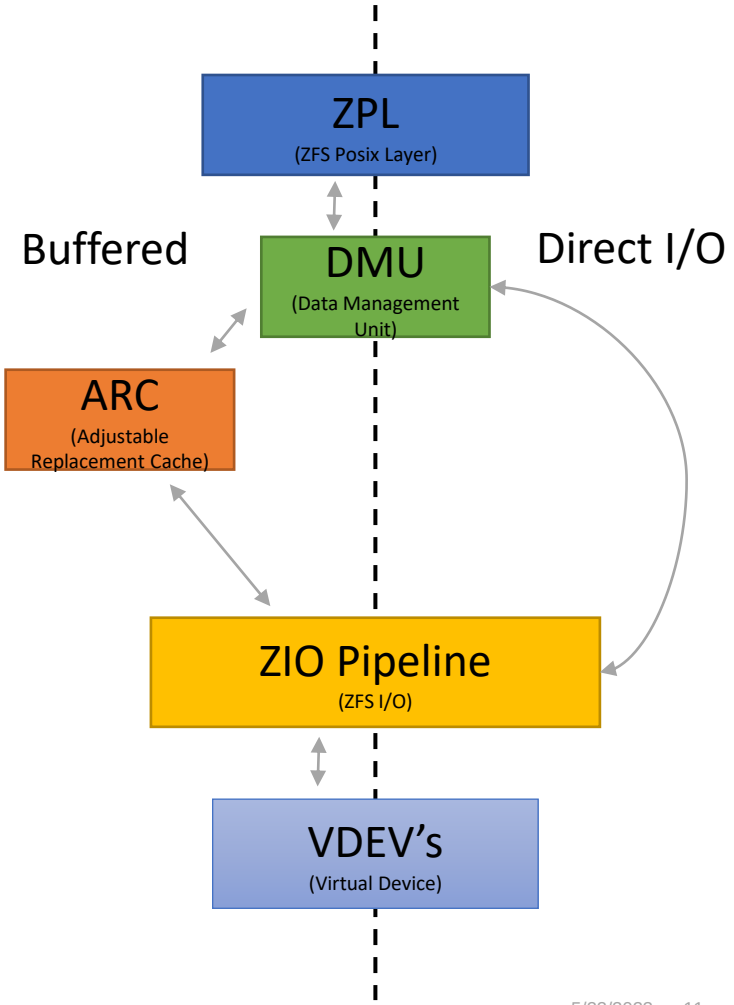
Motivation: Why would you want to bypass the ARC? Have we lost our minds?!

- Pass O_DIRECT flag in open() call
- From the Linux man page for open():
  - Try to minimize cache effects of I/O to and from this file… File I/O is done directly to/from user-space buffers.
- Loose rules around exact semantics
- Currently ZFS happily accepts the O_DIRECT flag but silently ignores it
- Direct I/O is not the solution to all problems (The ARC still important)
- Databases – Own caching mechanisms
- Writing large amounts of data not intending read (checkpoint)
- Reading large amounts of data once (restart)
- **Using low latency, high bandwidth devices (NVMe) as VDEVs**
  - Average 1.5 – 3x Speedup with Direct IO support added to ZFS

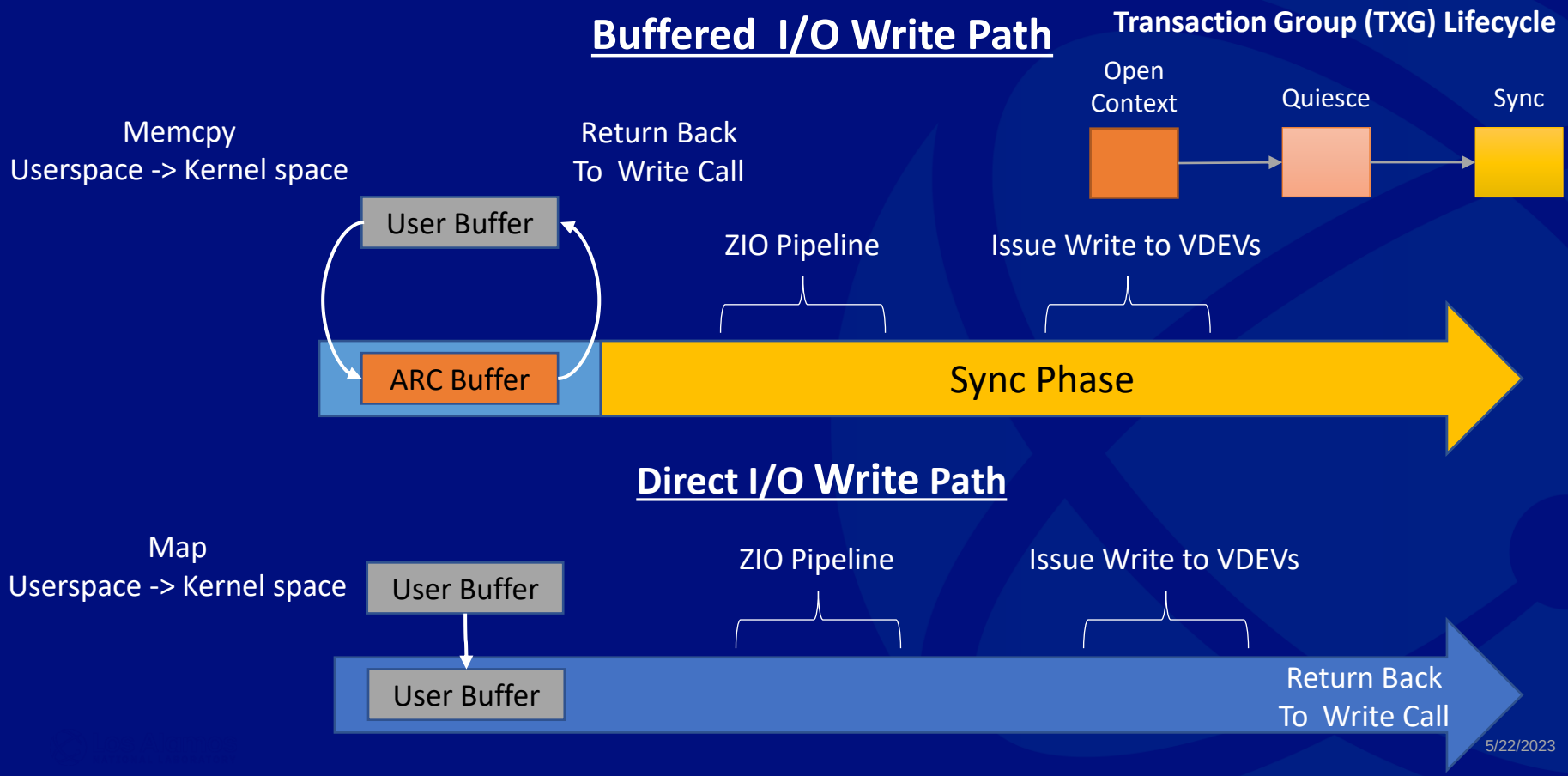
Implementation: Direct I/O Read in ZFS (Big Picture)

Buffered
- Cached? → Copy from ARC
- Issue to ZFS pipeline
  - Copy to ARC
  - Copy to user buffer

Direct I/O
- Bypass ARC
- User pages are directly mapped



Implementation: Direct I/O Write in ZFS (Big Picture)

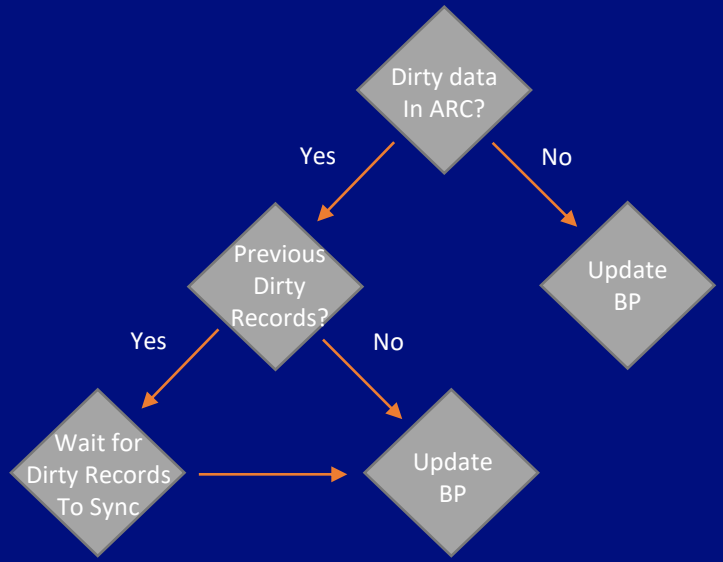


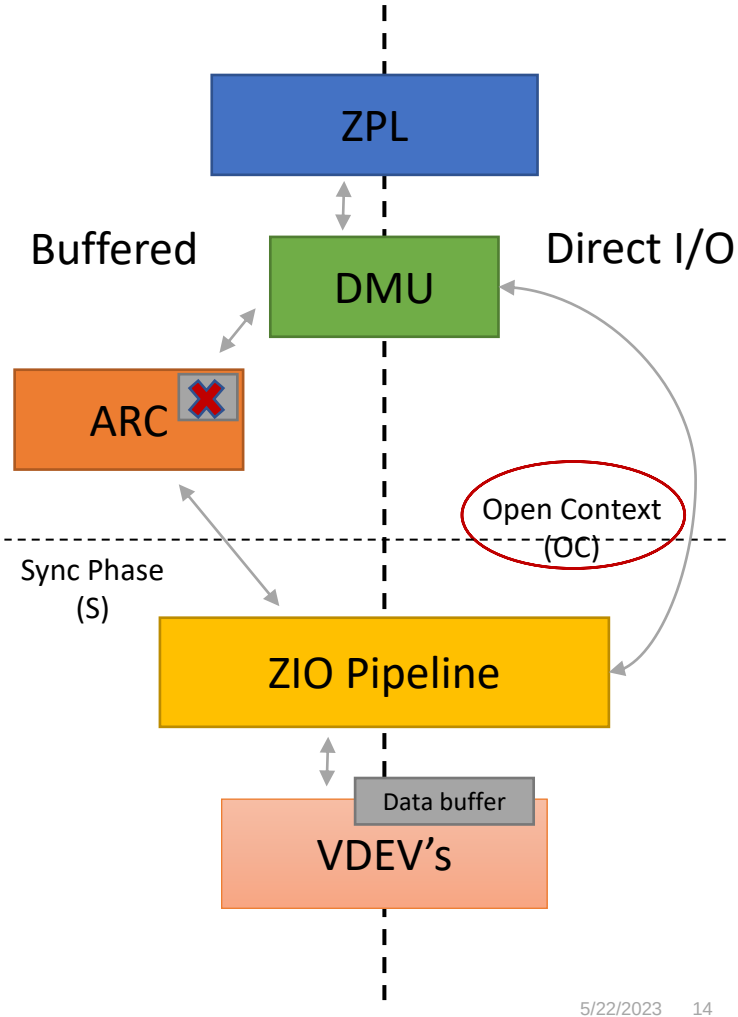
Implementation: General Details

- O_DIRECT does not imply O_SYNC
- Must be PAGE_SIZE aligned if using O_DIRECT flag
  - Returns EINVAL
- Mmap'ed files will **silently** ignore direct I/O and used buffered
- Direct I/O accounting:
  - Linux: /proc/spl/kstat/zfs/*poolname*/iostats
  - FreeBSD: sysctl kstat.zfs.*poolname*.misc.iostats
- Direct I/O requests are issued with sync priority
  - ZIO_PRIORITY_SYNC_{READ/WRITE}
- **ARC is coherent with O_DIRECT**

Caveat 1: Direct I/O Write

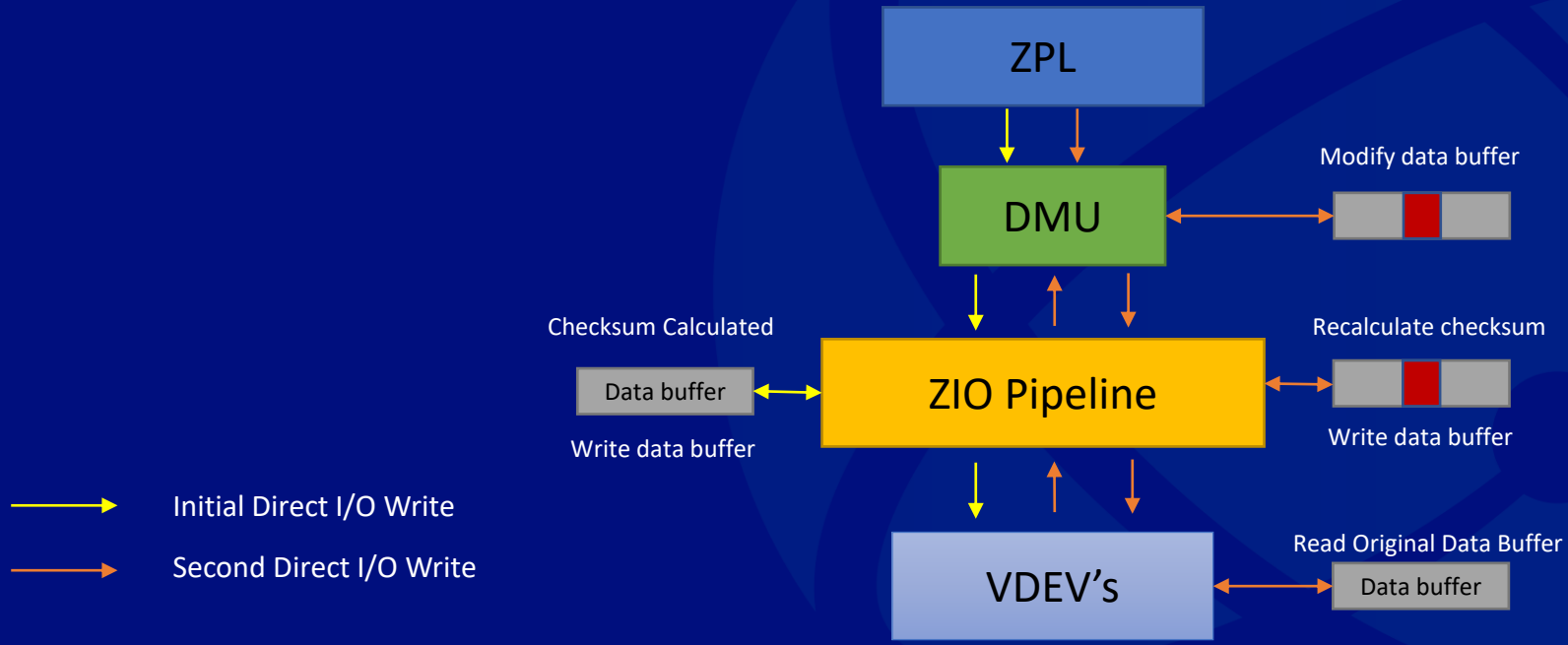
Stale data in ARC is removed or synced out (ARC Coherency)



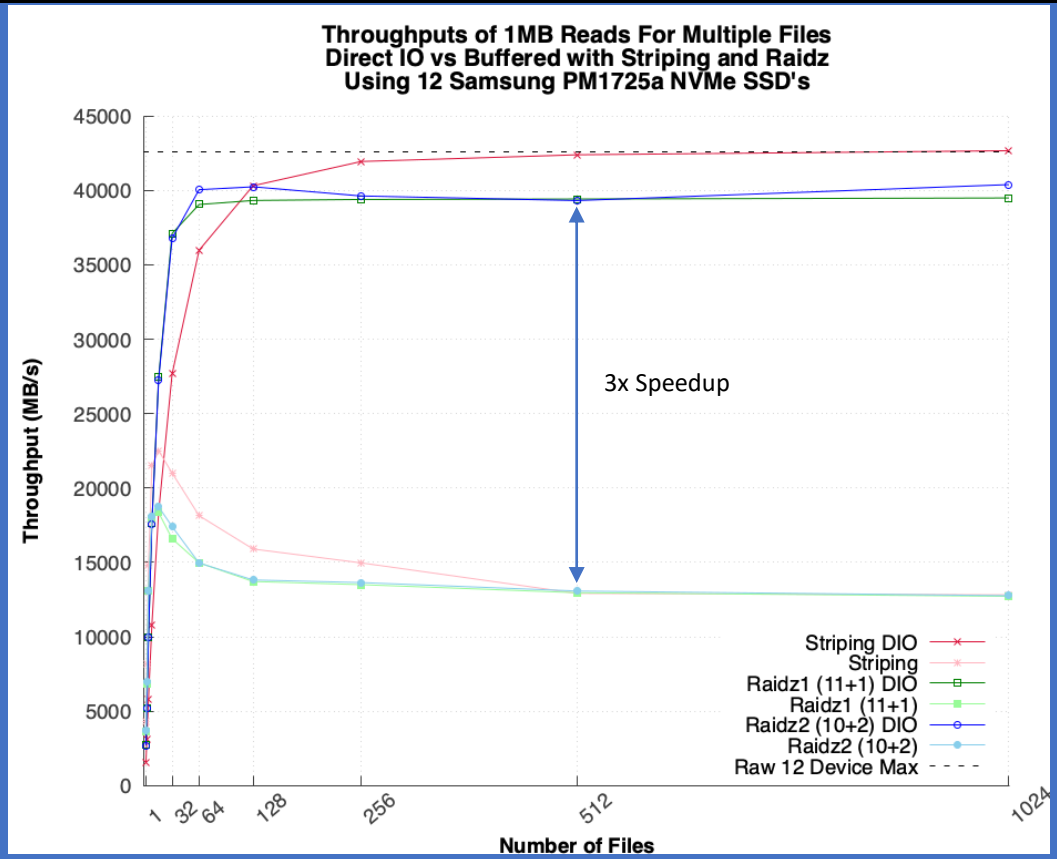


Caveat 2: Direct I/O Write Continued…

- Must be recordsize aligned to avoid Read/Write/Modify
  - If not O_DIRECT flag silently ignored

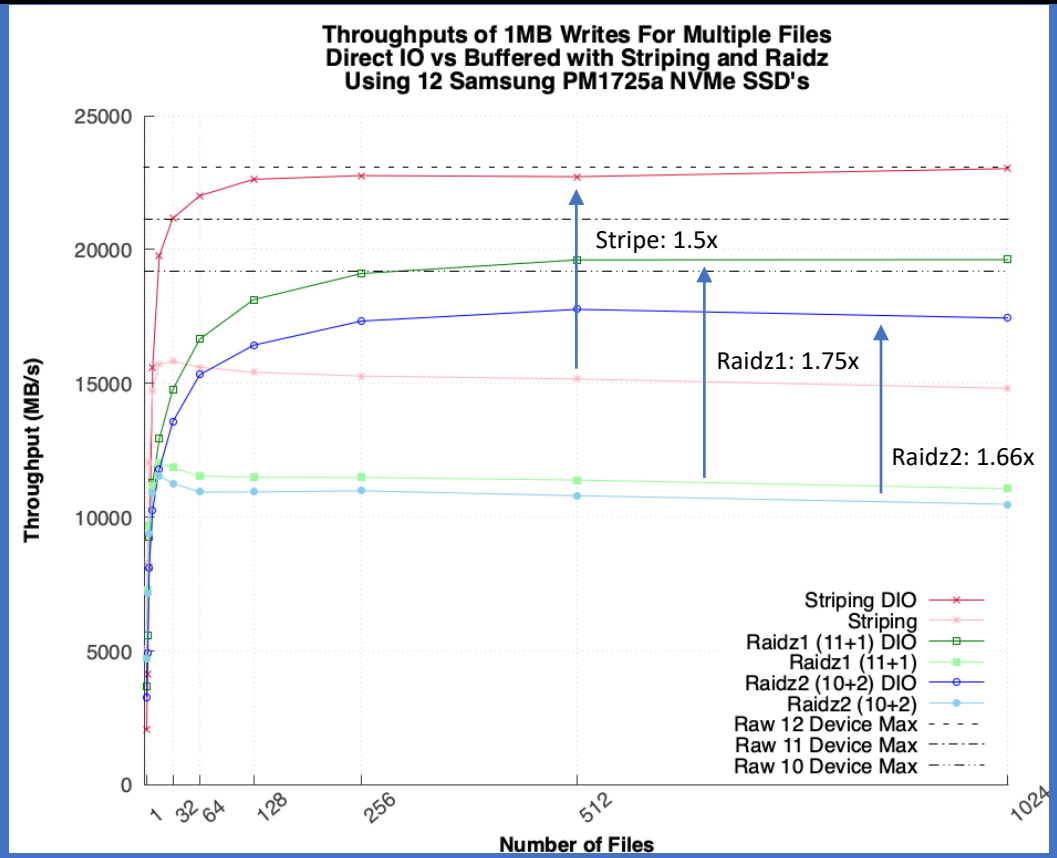


ZFS Seq. Read gen3 Performance Results



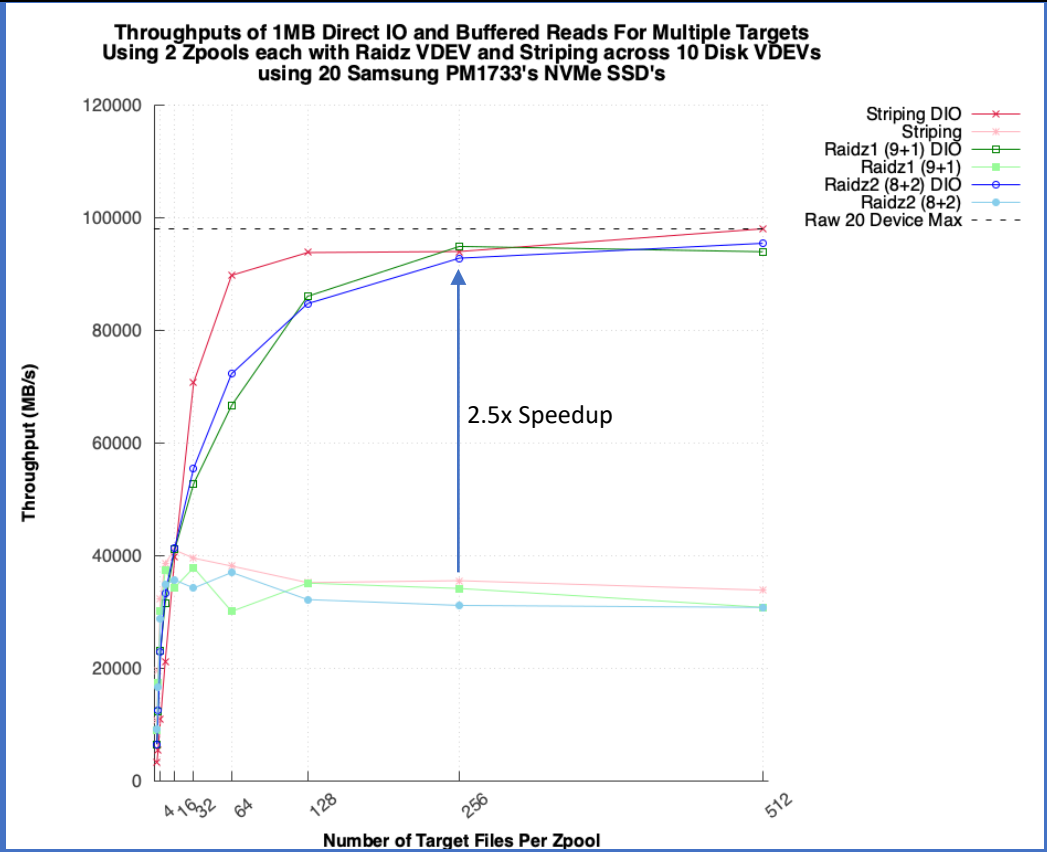
Stripe: 99% Raidz1: 92% Raidz2: 95%
Percentage Total Device Bandwidth with O_DIRECT

Seq. Write gen3 Performance Results



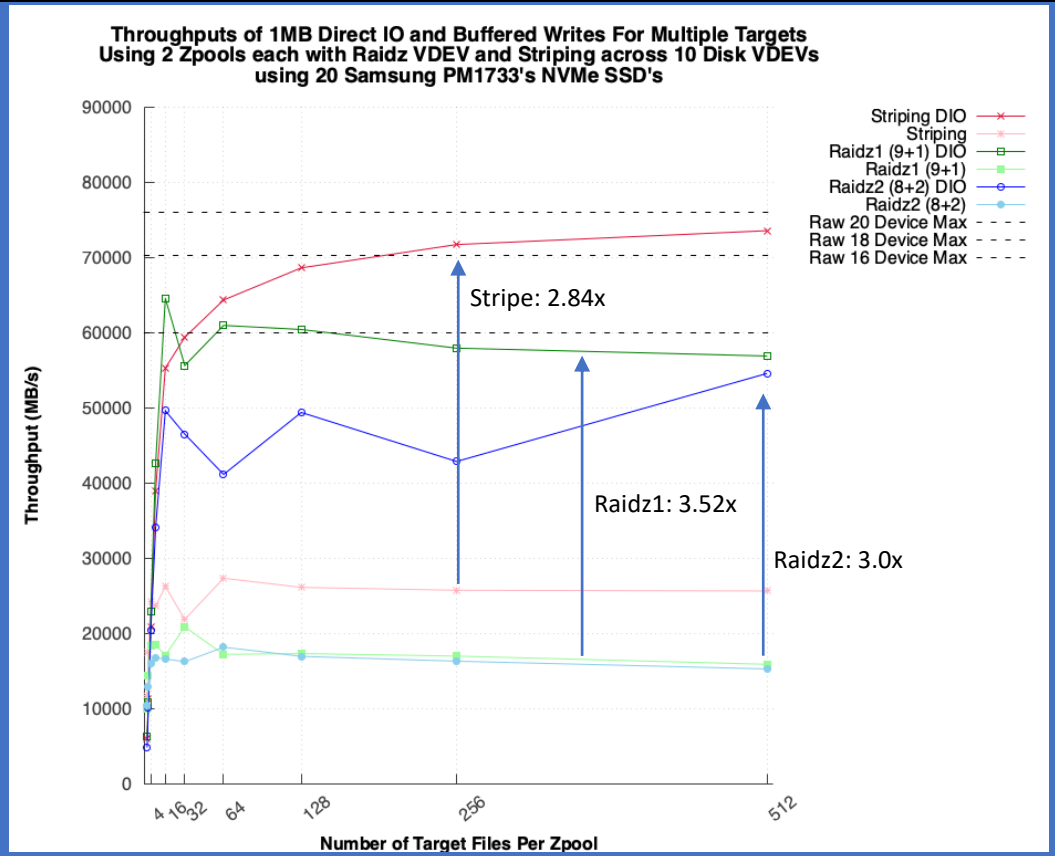
Stripe: 99% Raidz1: 93% Raidz2: 92%
Percentage Total Device Bandwidth with O_DIRECT

ZFS Seq. Read gen4 Performance Results



Stripe: 99% Raidz1: 97% Raidz2: 97%
Percentage Total Device Bandwidth with O_DIRECT

ZFS Seq. Write gen4 Performance Results



Stripe: 97% Raidz1: 92% Raidz2: 91%

Percentage Total Device Bandwidth with O_DIRECT

arcstat 1

```
[root@quilt00 ~]# arcstat 1
    time  read  ddread  ddh%  dmread  dmh%  pread  ph%   size     c  avail
12:16:37     0       0     0       0     0      0    0   2.9M   62G   116G
12:16:38     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:39     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:40     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:41     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:42     4       0     0       4   100      0    0   2.9M   62G   116G
12:16:43     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:44     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:45     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:46     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:47     4       0     0       4   100      0    0   2.9M   62G   116G
12:16:48     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:49     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:50     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:51     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:52     4       0     0       4   100      0    0   2.9M   62G   116G
12:16:53     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:54     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:55     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:56     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:57     4       0     0       4   100      0    0   2.9M   62G   116G
12:16:58     1       0     0       1   100      0    0   2.9M   62G   116G
12:16:59     1       0     0       1   100      0    0   2.9M   62G   116G
12:17:00     1       0     0       1   100      0    0   2.9M   62G   116G
12:17:01     1       0     0       1   100      0    0   2.9M   62G   116G
```

zpool iostat –qv 1

```
    nvme10n1     -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme11n1     -      -      0      0      0      0      0      0      0      0      0      0
------------  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  ---
                capacity     operations     bandwidth    syncq_read    syncq_write   asyncq_read   asy
pool          alloc   free   read  write   read  write   pend  activ   pend  activ   pend  activ   pe
------------  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----
local_zpool   1.37M  17.5T      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
  raidz2-0    1.37M  17.5T      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme0n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme1n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme2n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme3n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme4n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme5n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme6n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme7n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme8n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme9n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme10n1     -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme11n1     -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
------------  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----
                capacity     operations     bandwidth    syncq_read    syncq_write   asyncq_read  asyncq_write   scrubq_read   trimq_write  rebuildq_write
pool          alloc   free   read  write   read  write   pend  activ   pend  activ   pend  activ   pend  activ   pend  activ   pend  activ   pend  activ
------------  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----
local_zpool   1.37M  17.5T      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
  raidz2-0    1.37M  17.5T      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme0n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme1n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme2n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme3n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme4n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme5n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme6n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme7n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme8n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme9n1      -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme10n1     -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
    nvme11n1     -      -      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
------------  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----  -----
```

zpool status and FIO runs

```
[root@quilt00 fio_runs]# ./run_fio_jobs.sh
```

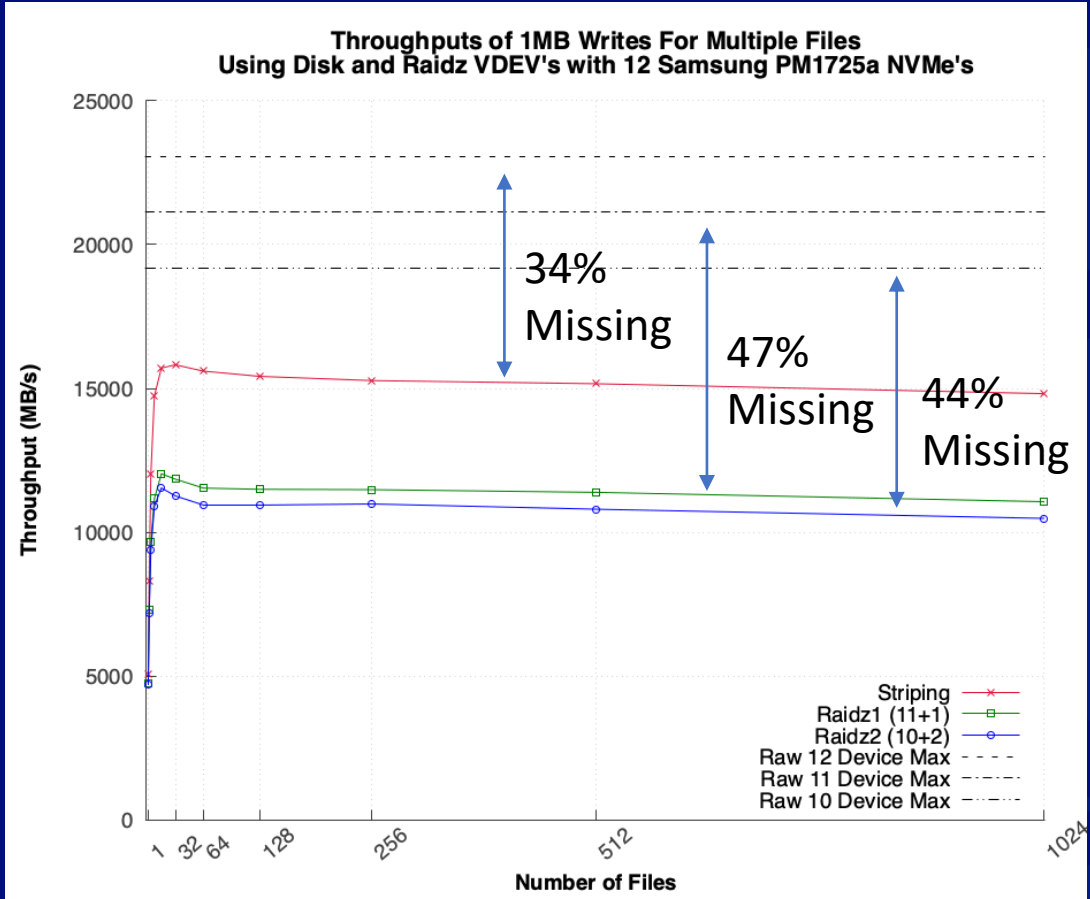
zpool iostats

```
Every 1.0s: cat /proc/spl/kstat/zfs/local_zpool/iostats          quilt00: Wed May 10 12:17:01 2023

125 1 0x01 26 7072 77419421348871 77481411585237
name                            type data
trim_extents_written            4    0
trim_bytes_written              4    0
trim_extents_skipped            4    0
trim_bytes_skipped              4    0
trim_extents_failed             4    0
trim_bytes_failed               4    0
autotrim_extents_written        4    0
autotrim_bytes_written          4    0
autotrim_extents_skipped        4    0
autotrim_bytes_skipped          4    0
autotrim_extents_failed         4    0
autotrim_bytes_failed           4    0
simple_trim_extents_written     4    0
simple_trim_bytes_written       4    0
simple_trim_extents_skipped     4    0
simple_trim_bytes_skipped       4    0
simple_trim_extents_failed      4    0
simple_trim_bytes_failed        4    0
arc_read_count                  4    0
arc_read_bytes                  4    0
arc_write_count                 4    247
arc_write_bytes                 4    62880
direct_read_count               4    0
direct_read_bytes               4    0
direct_write_count              4    0
direct_write_bytes              4    0
```

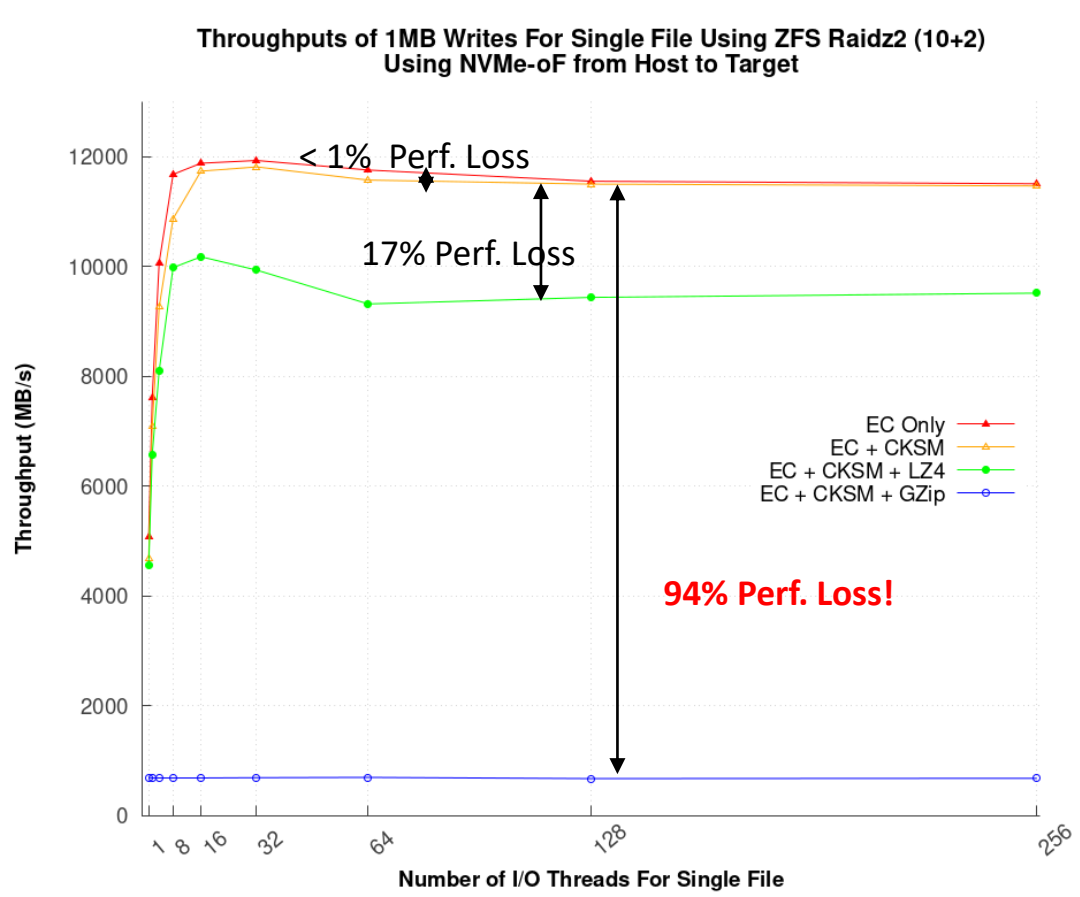
```
[zfs] 0:bash*                                                    "quilt00" 12:16 10-May-23
```

Computational Storage Offloads for OpenZFS

ZFS gen3 NVMe Zpools Sequential Write Performance



Stock ZFS Performance with Various Features Enabled



What can we do to improve performance?

- Use computational storage to offload operations
  - Perform operations that are CPU/memory bandwidth intensive when run on host
  - Can be implemented with FPGAs
  - Data Processing Unit (DPU)

Eideticom Noload Computational Storage Processor (CSP)

Doesn’t ZFS already support offloading?

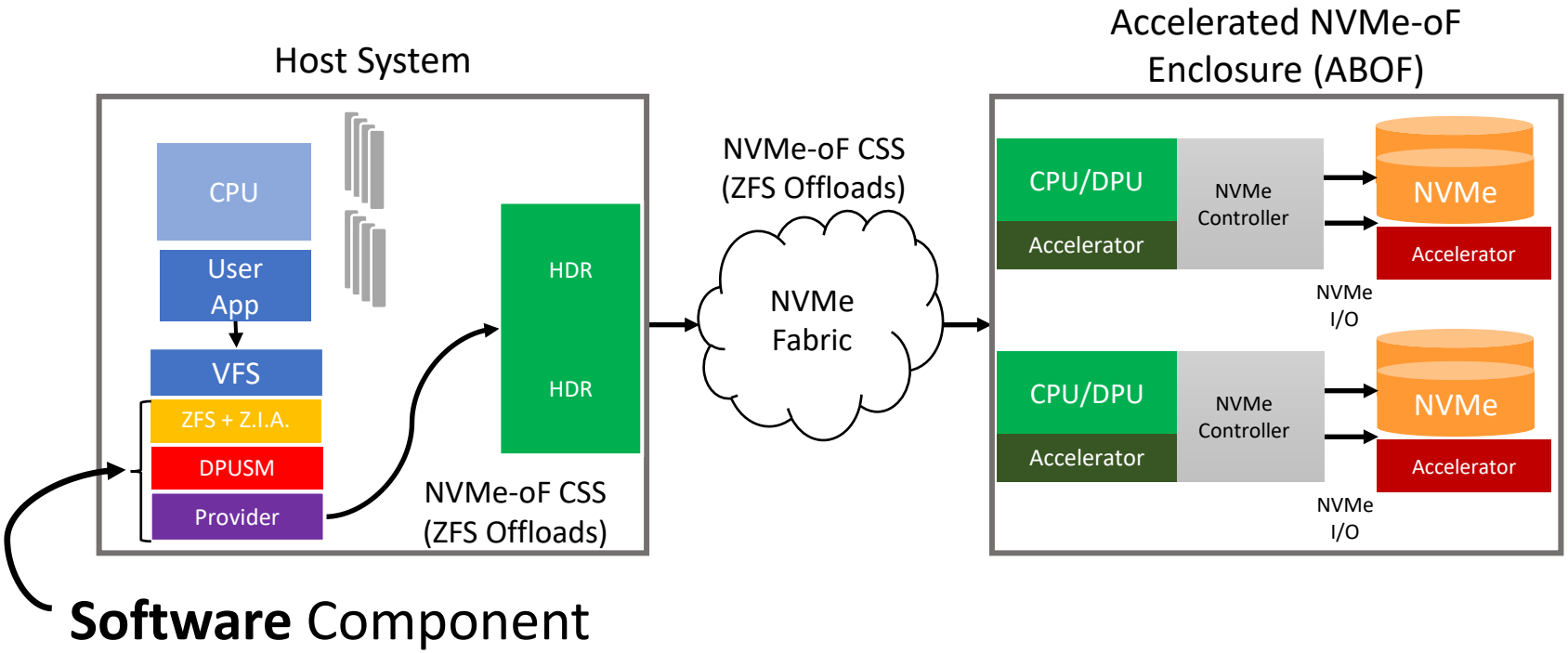
- Intel® QuickAssist Technology (Intel® QAT)
  - **Doesn’t work on AMD machines**

- Requires ZFS to be reconfigured

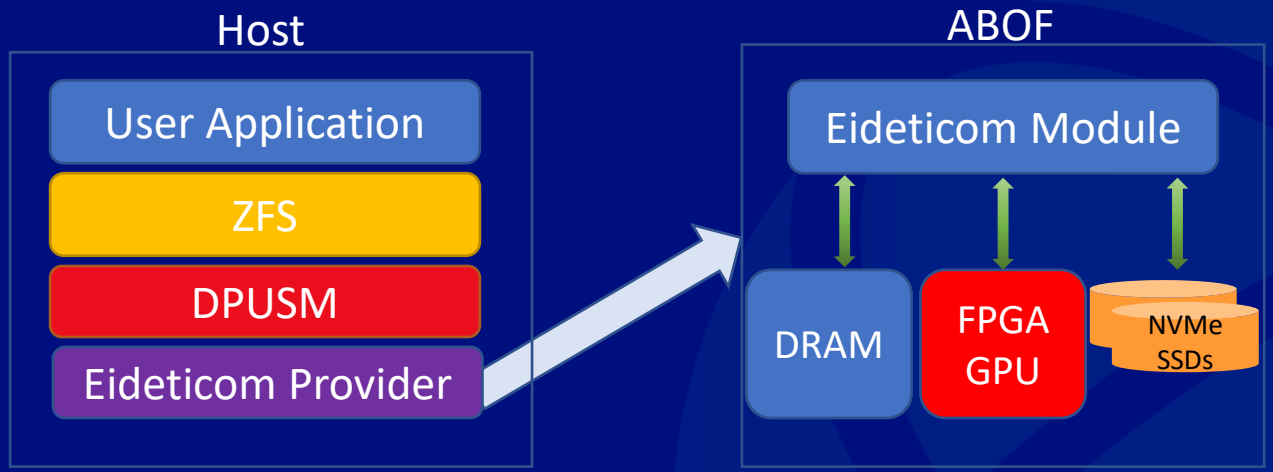
- Each offload operation is done independently of each other
  - Encryption – AES-GCM
  - Compression – GZIP
  - Checksum – SHA256

- Not extensible
  - API updates need to be merged upstream

Accelerated ZFS with Disaggregated Storage

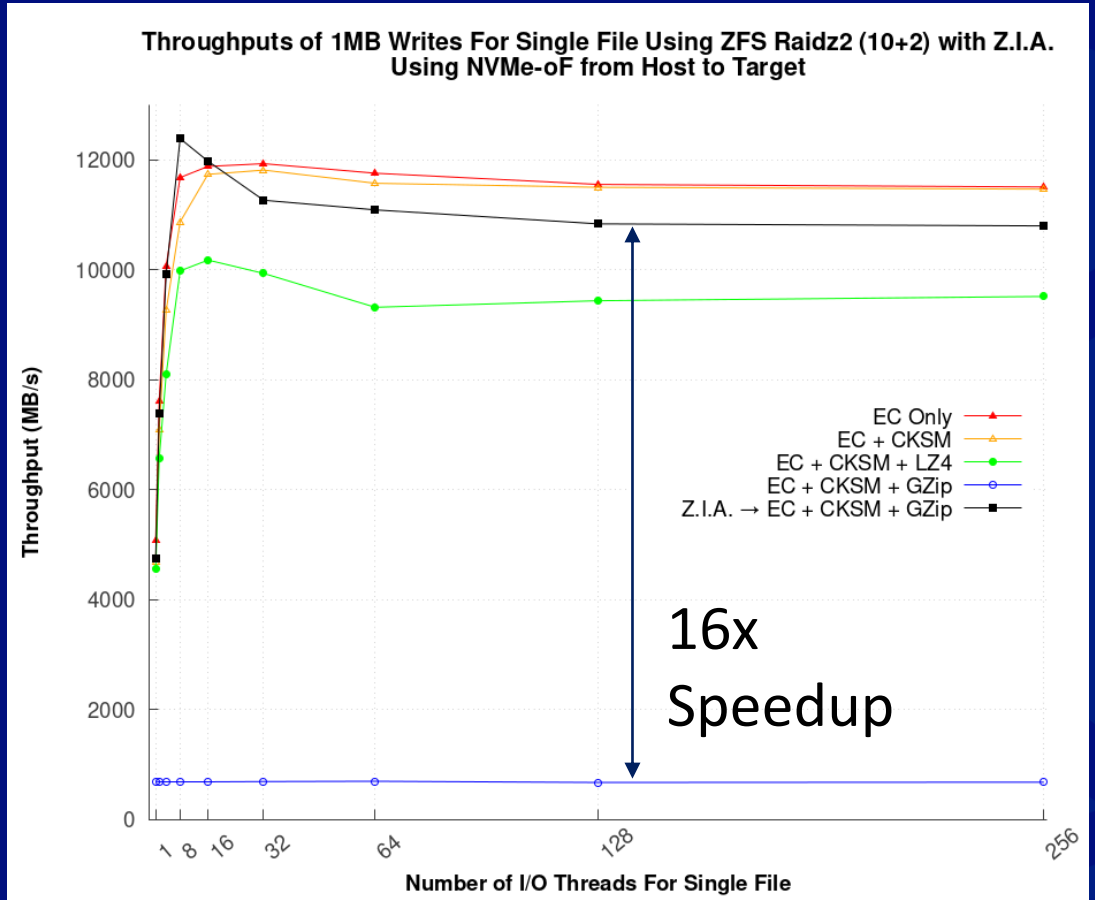


Target side kernel module



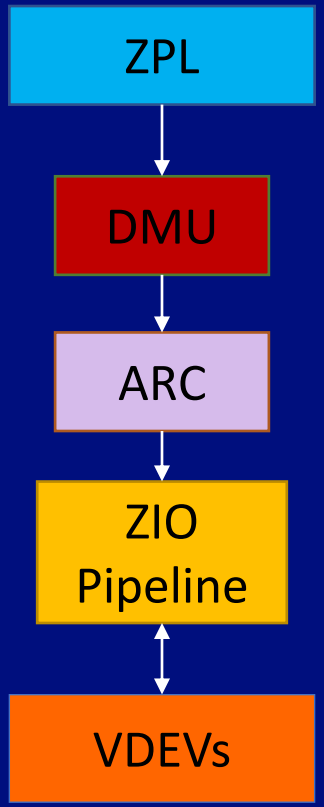
- Tie into NVMe-oF module
- Use a set of vendor op codes to
  - create/free buffer
  - load/store buffer from disk
  - read/write buffer
  - Perform operation on buffer
    - (compress/decompress/checksum/EC)

Z.I.A. Performance with Eideticom NoLoad CSP



ZFS Interface for Accelerators
+
Data Processing Unit Services Module

ZFS Write Pipeline



ZFS POSIX layer

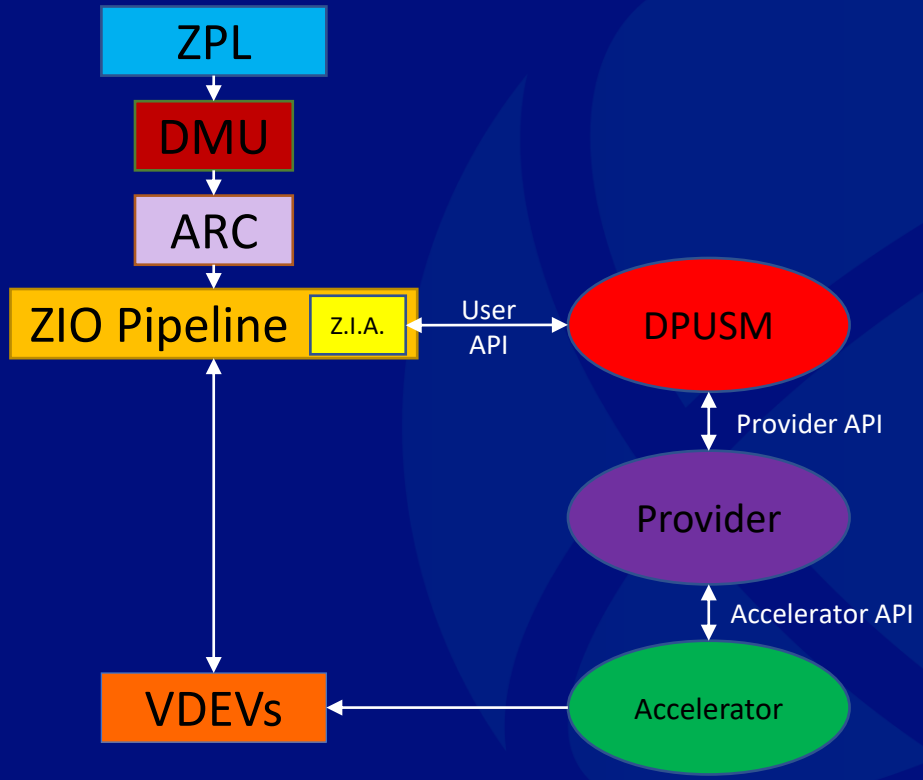
Data Management Unit

Adjustable Replacement Cache

ZFS I/O Pipeline

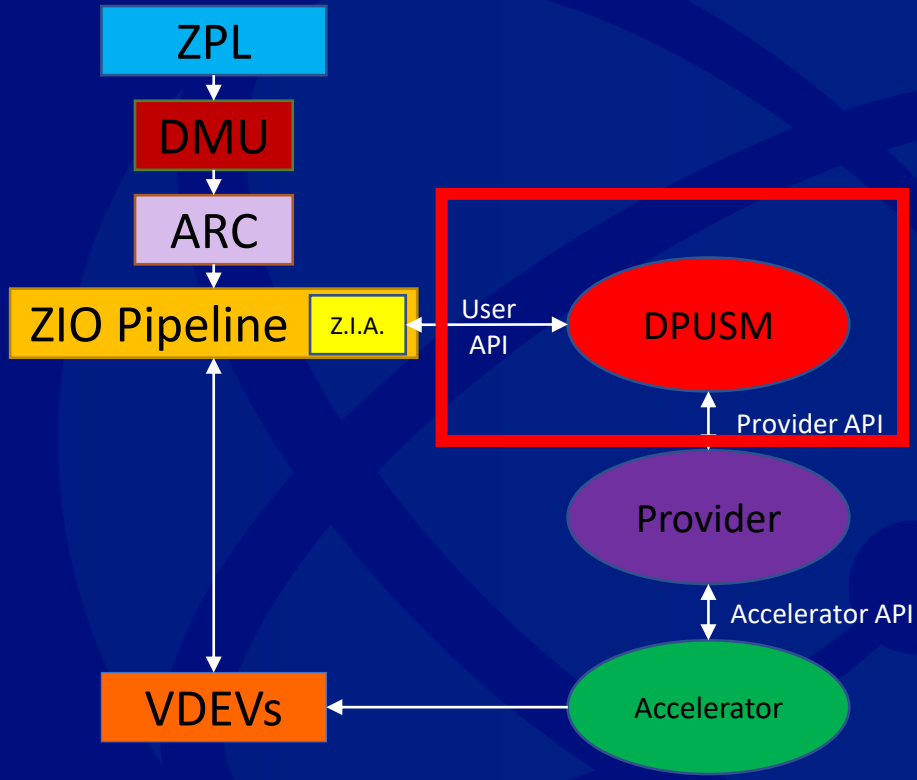
Virtual Devices

Z.I.A. Write Pipeline



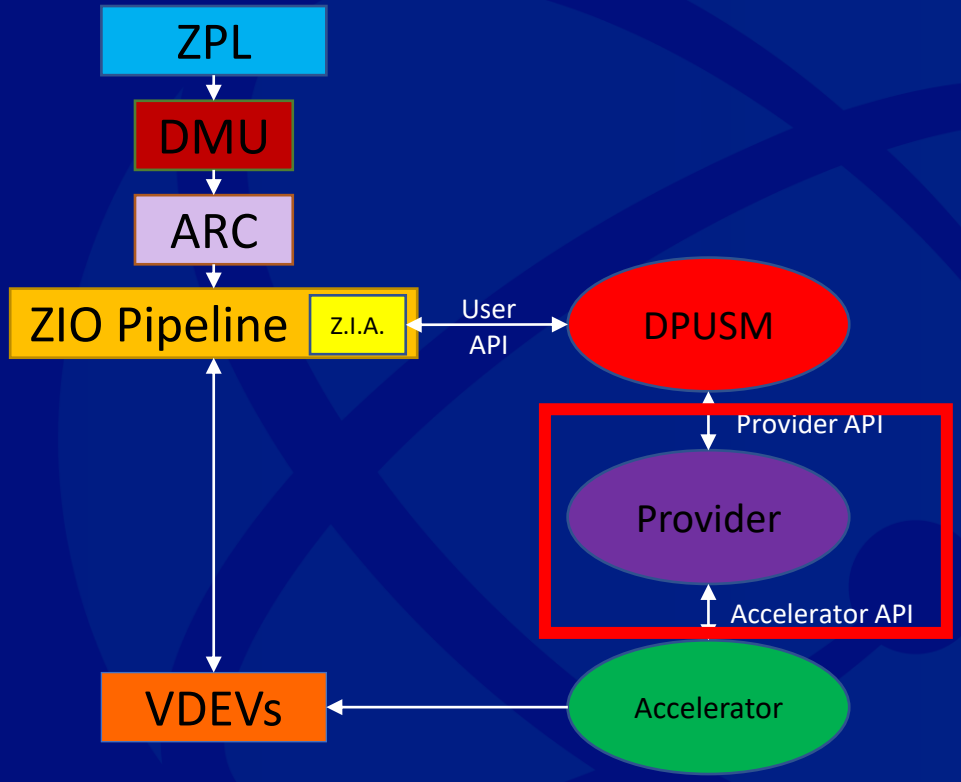
Data Processing Unit Services Module (DPUSM)

- Kernel module
- Standardized APIs for leveraging computational storage
  - Provider API
  - User API
- Acts as registry for providers



Providers

- Kernel module
- DPUSM wrapper for accelerator specific code
- Does not know anything about user
- Declares what the accelerator provides
- Usually implemented by accelerator vendor



Provider Implementation Basics

- `#include <accelerator_header.h>`
- `#include <dpusm/provider_api.h>`
- Fill in DPUSM provider functions struct
  - Analogous to VFS function pointers
- Register provider with DPUSM on module initialization

1. Give user handle that references accelerator memory
2. Get user (in-memory) data into accelerator (copy, dma, etc.) via handles
3. Accept handles for operations

- Communication with accelerator is connection protocol agnostic

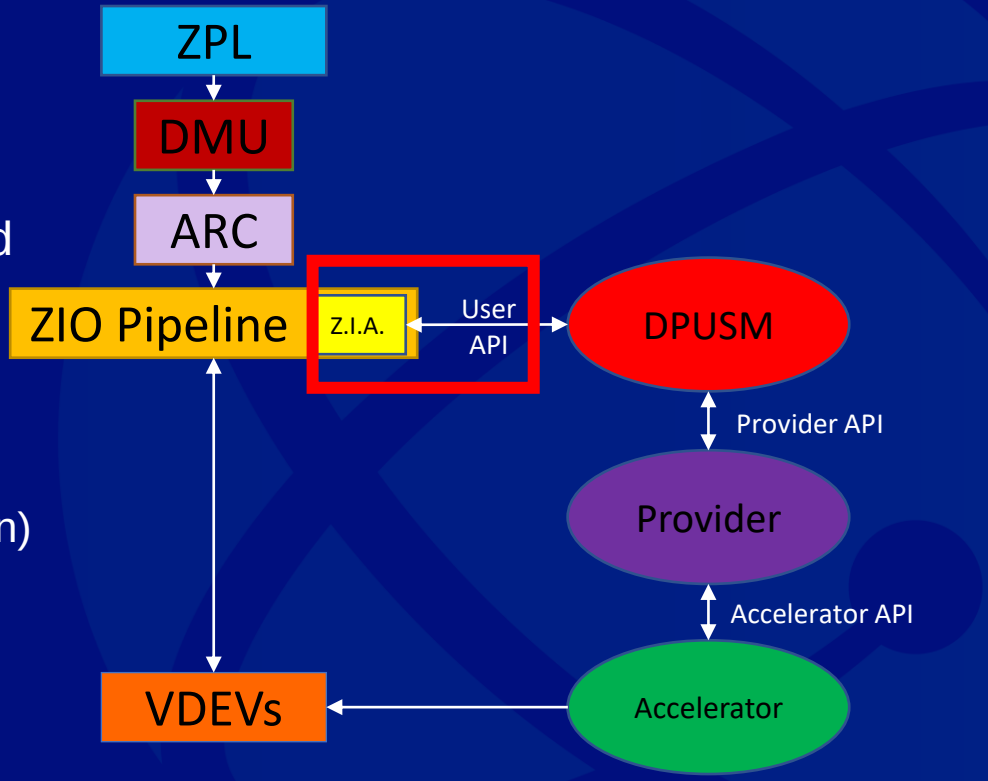
Using a Provider

- `#include <dpusm/user_api.h>`
- Find provider
- Use provider functions in DPUSM user functions struct

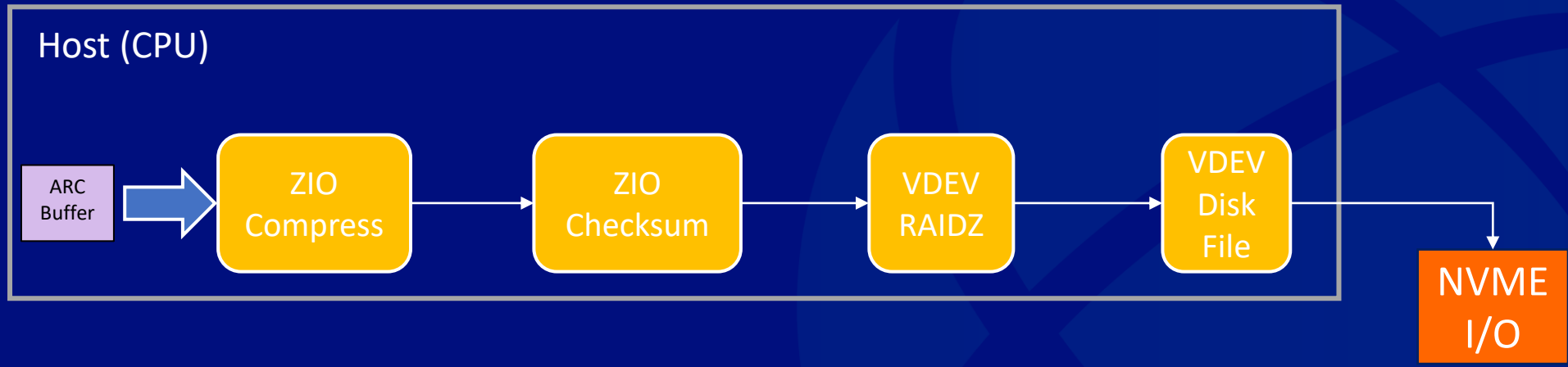
1. Get opaque handle (`void *`) to accelerator memory (wrapped by provider)
2. Get in-memory data to accelerator via handle
3. Pass handle(s) to provider functions to operate on data

ZFS Interface for Accelerators (Z.I.A.)

- Modifications to the ZFS **write** pipeline
- Transparent acceleration of CPU and memory intensive ZFS write operations with accelerators
  - Compression
  - Checksum
  - RAIDZ (Generation and Reconstruction)
  - I/O
- User data access not affected
  - During write
  - Afterwards



ZFS Write Pipeline

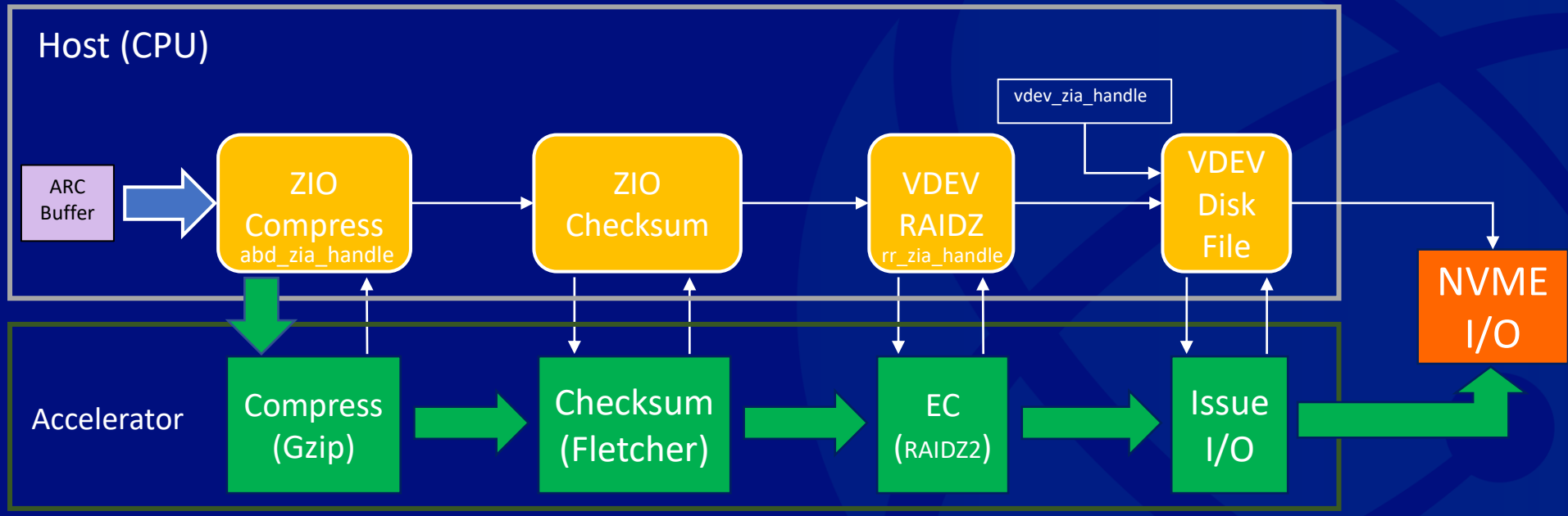


General Description of Changes

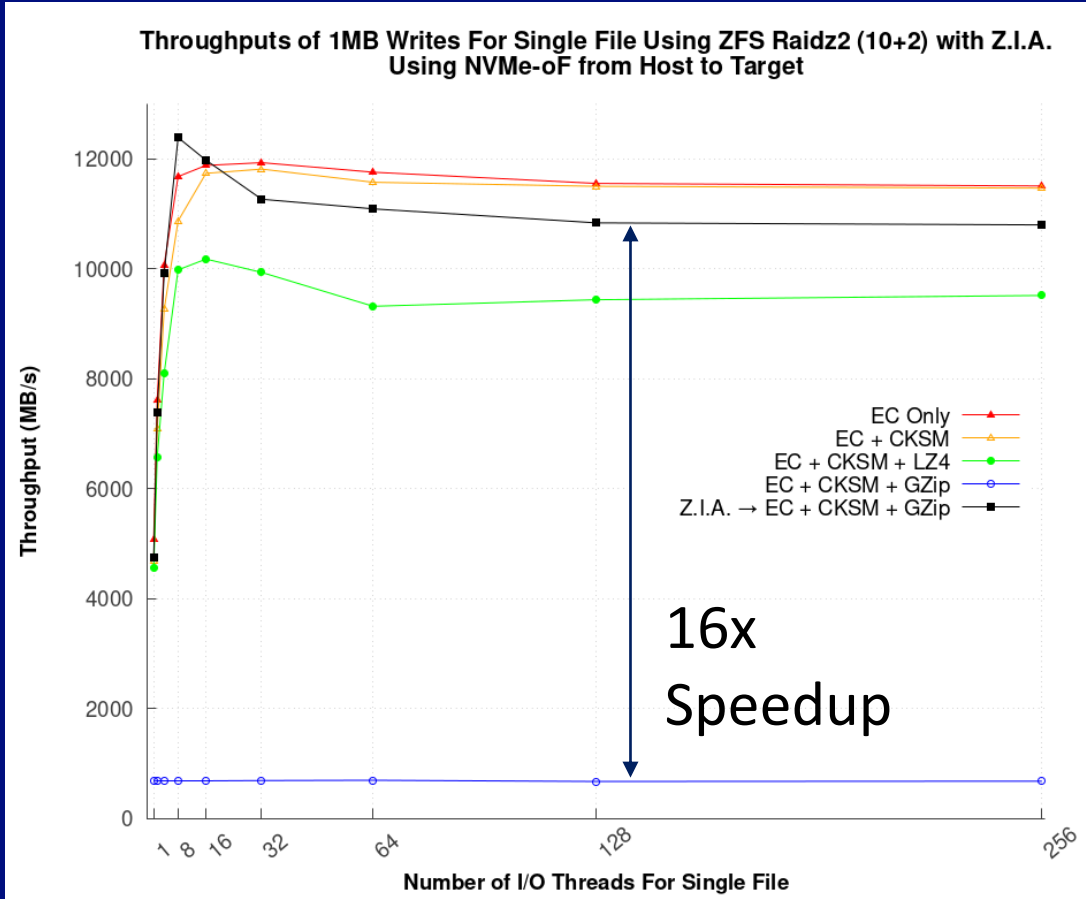
- If data is not offloaded at start of stage, offload it
- Run the operation
- Return status code (not data)

- If Z.I.A. fails, bring data back to memory, fall back to running operation in software
- If offloaded data cannot be returned to memory, restart write pipeline
  - A copy of the original data is still available in ZFS

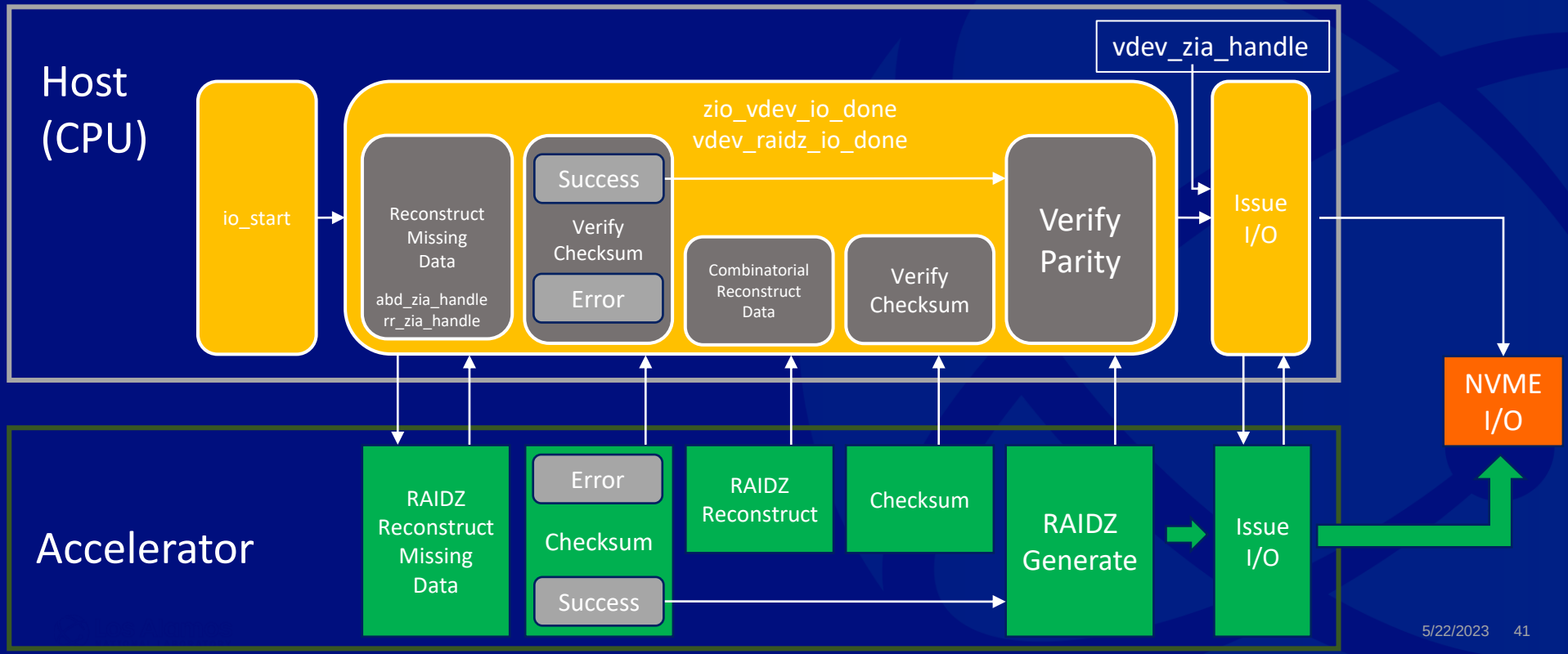
Z.I.A. Write Pipeline



Z.I.A. Performance with Eideticom NoLoad CSP



Z.I.A. (RAIDZ) Resilver



Get Z.I.A.

- Z.I.A. Pull Request
  - https://github.com/openzfs/zfs/pull/13628
  - Comes with software provider that links back into ZFS

- Data Processing Unit Services Module
  - https://github.com/hpc/dpusm

Z.I.A. Demo

Backing Drives

- Direct attached
- NVMe-oF

DPUSM Setup

- `git clone https://github.com/hpc/dpusm.git`
- `cd dpusm`
- `make`
- `sudo insmod dpusm.ko`

Compile ZFS + Z.I.A.

- `git clone -b zia https://github.com/hpc/zfs.git`
- `cd zfs`
- `./autogen.sh`
- `./configure --with-zia=${HOME}/dpusm`
- `make -j`

Build and Load Provider

- Should not link with ZFS, so can build and load after ZFS is loaded

- Z.I.A. software provider is a special case
  - Depends on ZFS
  - Builds with Z.I.A.
  - Load after loading ZFS
    - module/zia-software-provider.ko

Set up zpool

- Load ZFS
  - `scripts/zfs.sh`
- Limit ARC size
  - `echo 17179869184 > /sys/module/zfs/parameters/zfs_arc_max`
- Create zpool
  - `zpool create -o ashift=12 local_zpool raidz2 /dev/nvme0n{1..12}`
- Set up zpool properties
  - `zfs set recordsize=1M local_zpool`
  - `zfs set compression=gzip local_zpool`
  - `zfs set checksum=fletcher4 local_zpool`

Write Enough Data to Force Memory Pressure on ARC

- `fio zia_demo.fio`

- 16 target files x 4GB
  - 4x ARC size to force eviction

- ~1GB/s
  - Bottleneck is memory bandwidth due to compression, not I/O

```
[global]
name=zia_demo
direct=0
ioengine=psync
bs=1m
size=4g
fallocate=0
rw=write
buffer_compress_percentage=25

[job0]
filename=/local_zpool/file0

[job1]

…
```

Enable Offloading with Z.I.A.

- `zpool set zia_provider="athena_example_provider" local_zpool`

- `zpool set zia_compress="on" local_zpool`
- `zpool set zia_checksum="on" local_zpool`
- `zpool set zia_raidz2_gen="on" local_zpool`
- `zpool set zia_disk_write="on" local_zpool`

```
zia_compress
zia_decompress
zia_checksum
zia_raidz1_gen
zia_raidz2_gen
zia_raidz3_gen
zia_raidz1_rec
zia_raidz2_rec
zia_raidz3_rec
zia_file_write
zia_disk_write
```

Write Enough Data to Force ARC Flush (Again)

- `fio zia_demo.fio`
- Much Faster!

Come See Our Technical Talks!

- Tuesday, May 23, 1:00 – 3:00
  - Accelerated Disks and Flashes: LANL's early experience in Speeding Up Analytics Workloads Using Smart Devices
    - Qing Zheng
  - Computational Storage Solutions Over Fabrics for ZFS
    - Kelly Ursenbach

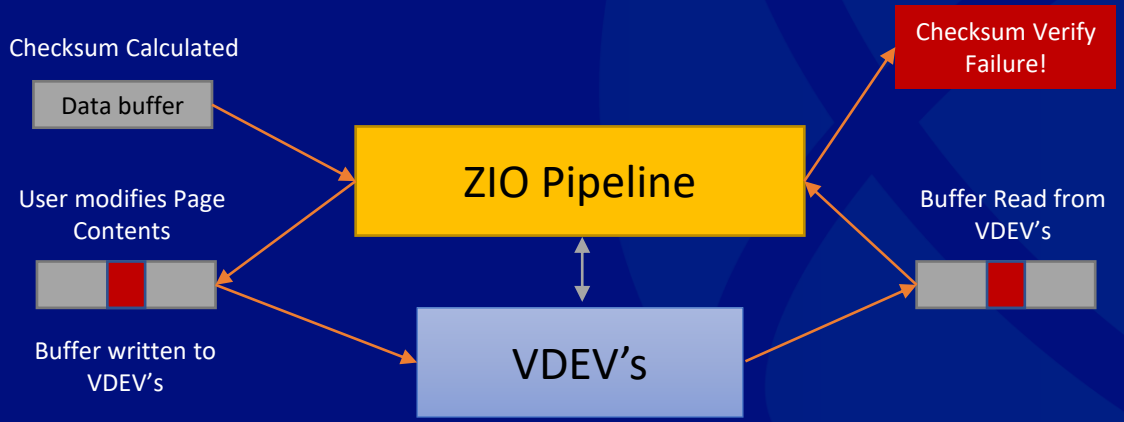
- Wednesday, May 24
  - 1:00 – 3:00
    - MarFS as a Multi-Level Erasure Archive
      - Garrett Ransom
  - 3:30 – 4:30
    - DNA Storage Erasure Encoding
      - Dominic Manno

- Questions?
- Comments?
- Test our pull requests

Addendum Slides

Implementation: Direct I/O Write Continued…

- Guard against users manipulating page data during write



Implementation: Direct I/O Write Continued…

- Added new module parameter zfs_vdev_direct_write_verify_pct
  - Required because anonymous pages on Linux can not be placed under write protection
  - Percentage of O_DIRECT writes to checksum verify before updating BP
  - Default set to 2%
  - zpool status –d can show if any O_DIRECT write checksum verify failures exist
  - Also logs ZED event dio_verify

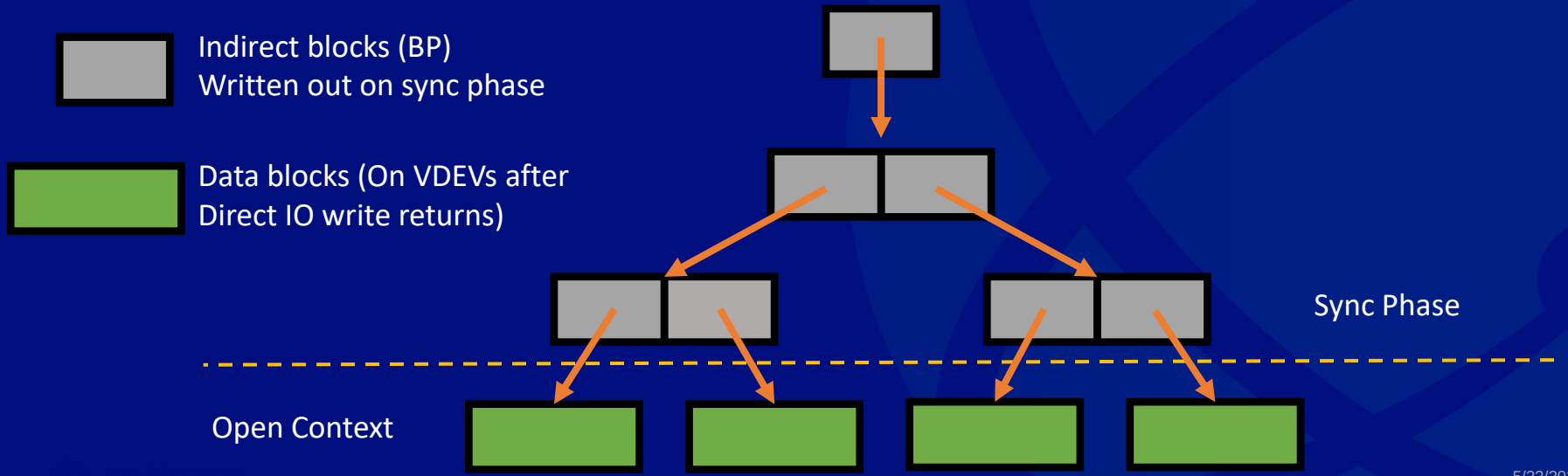
```
  pool: local_zpool
 state: ONLINE
config:

        NAME         STATE     READ WRITE CKSUM    DIO
        local_zpool  ONLINE       0     0     0      0
          raidz1-0   ONLINE       0     0     0      0
            nvme0n1  ONLINE       0     0     0      0
            nvme1n1  ONLINE       0     0     0      0
            nvme2n1  ONLINE       0     0     0      0

errors: No known data errors
```

Implementation: One last Note on Direct I/O in ZFS

- New dataset property added
  - direct=standard (default)
    - Follows semantics outlined so far
  - direct=always
    - Treat every read/write I/O request as Direct I/O (best effort)
    - However if not PAGE_SIZE aligned fall back to buffered
  - direct=disabled
    - Silently ignore O_DIRECT

Implementation: O_DIRECT vs O_SYNC

- O_DIRECT does not imply O_SYNC
  - Direct IO writes are written immediately to VDEVs
- O_SYNC will commit Direct IO block pointers (BP) to ZIL



Z.I.A. Usage (Admins)

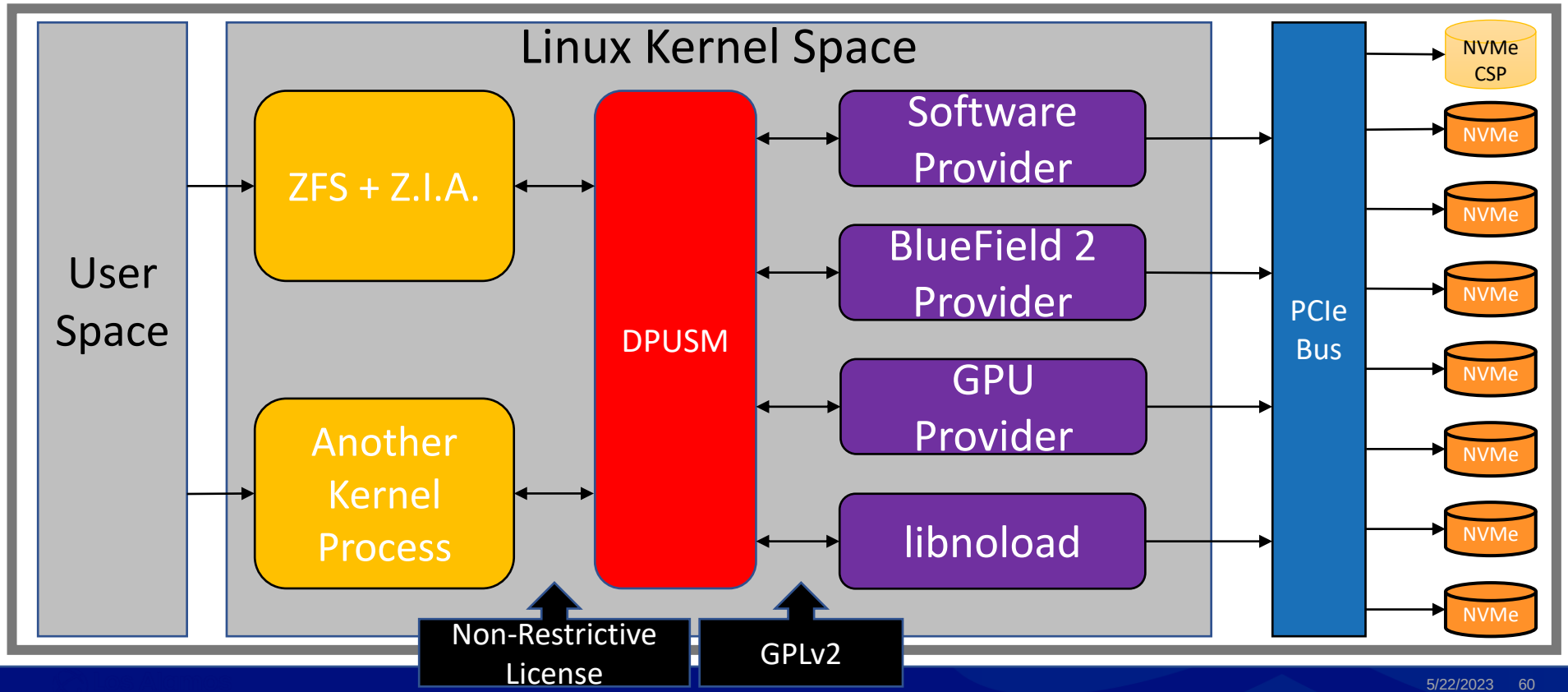
- Currently need to reconfigure ZFS with `--with-zia=<DPUSM Root>`
  - Expect that ZFS will always compile Z.I.A. once merged
  - Z.I.A. will not cause issues if DPUSM is not found at load time

- Select a provider
  - `zpool set zia_provider=<provider name> <zpool>`

- Enable offloading
  - `zpool set zia_<property>=on <zpool>`
  - Offloading only occurs if the ZFS stage is enabled

Accelerated ZFS with Converged Storage

EIDETICOM



Status of O_DIRECT on OpenZFS master

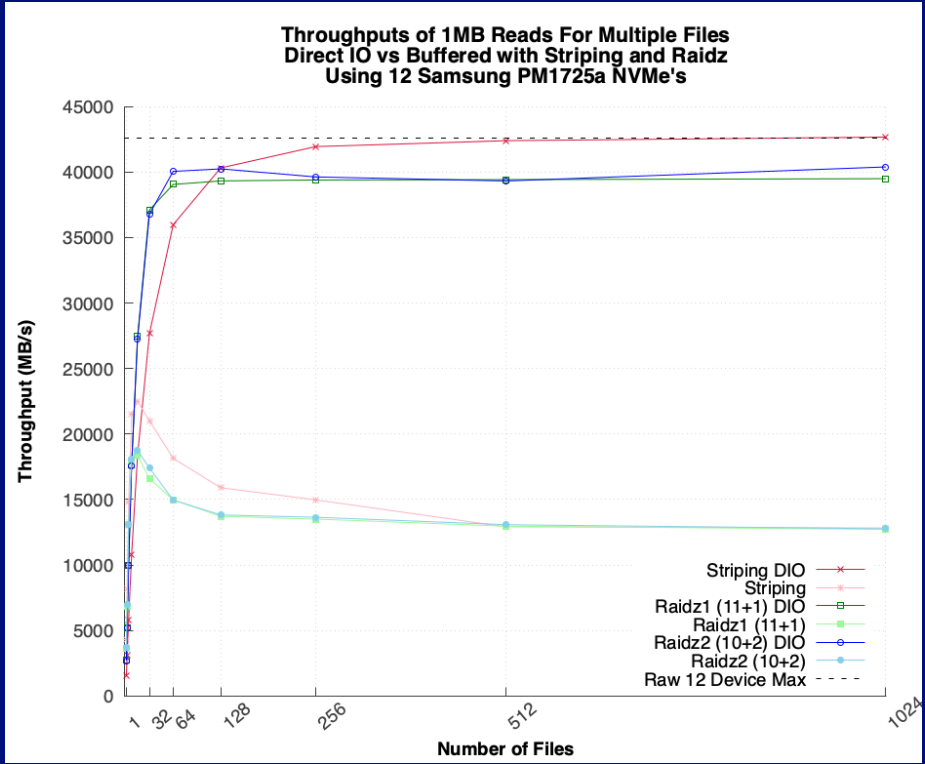
- Open PR on OpenZFS master (#10018)
  - https://github.com/openzfs/zfs/pull/10018
- Needs further code reviews
- FreeBSD performance testing
- Direct I/O for ZVOLS
- Aiming for inclusion in major point release (December 2023)

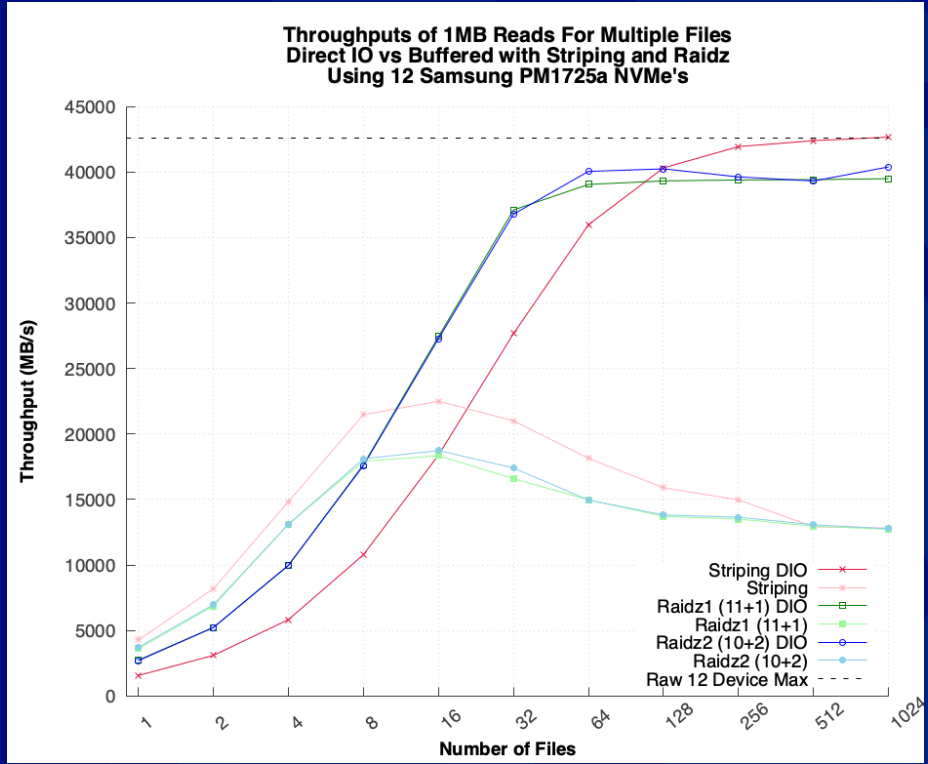
Performance Results: NVMe Zpool Test Configuration

- Server
  - Gigabyte R272-Z32-00
  - Single Socket: AMD EPYC Rome 7502 32c (Hyperthreading 64c)
  - 128 GB RAM, 8x 16 GB 2933 MT/s DDR4
  - CentOS 8.2, Kernel: 4.18.0-193.el8.x84_64
  - 12 Samsung PM1725a 1.6 TB
- IO Workload
  - Request block size 1M
  - 2 TB of total data read/written to Zpool
  - Sequential Read/Write, 1 – 1024 Files

- ZFS Settings/Configuration
  - ZFS recordsize=1M
  - zfs_vdev_sync_{read/write}_max_active=64
  - zfs_vdev_async_write_max_active=64
  - 1 Zpool 12 NVMe Disk VDEVs (Striping)
  - 1 Zpool 1x Raidz1 (11+1) VDEV
  - 1 Zpool 1x Raidz2 (10+2) VDEV
  - 1 Zpool 1x dRAID1 (11d:12c) VDEV
  - 1 Zpool 1x dRAID2 (10d:12c) VDEV

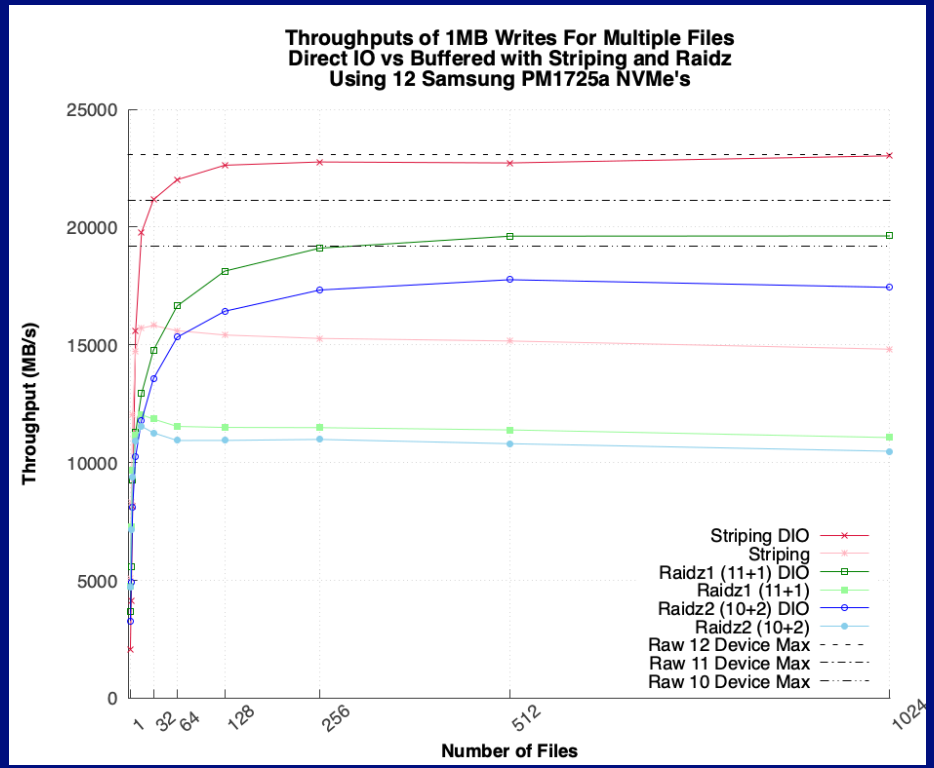
Gen3 Seq. Read Performance Results: Raidz

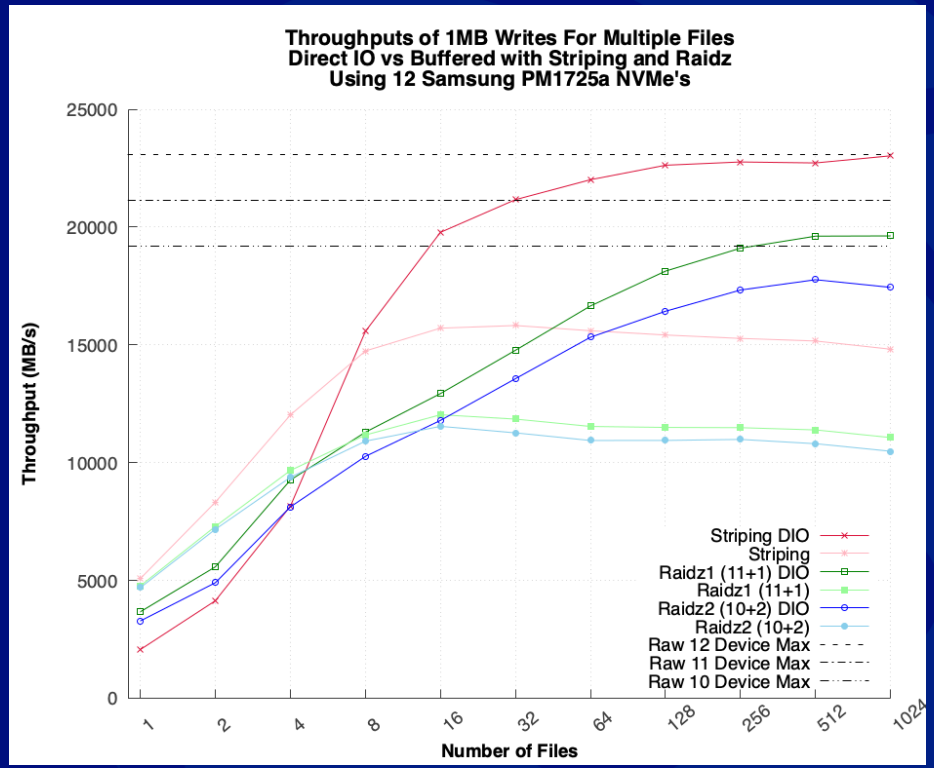




Log Scale

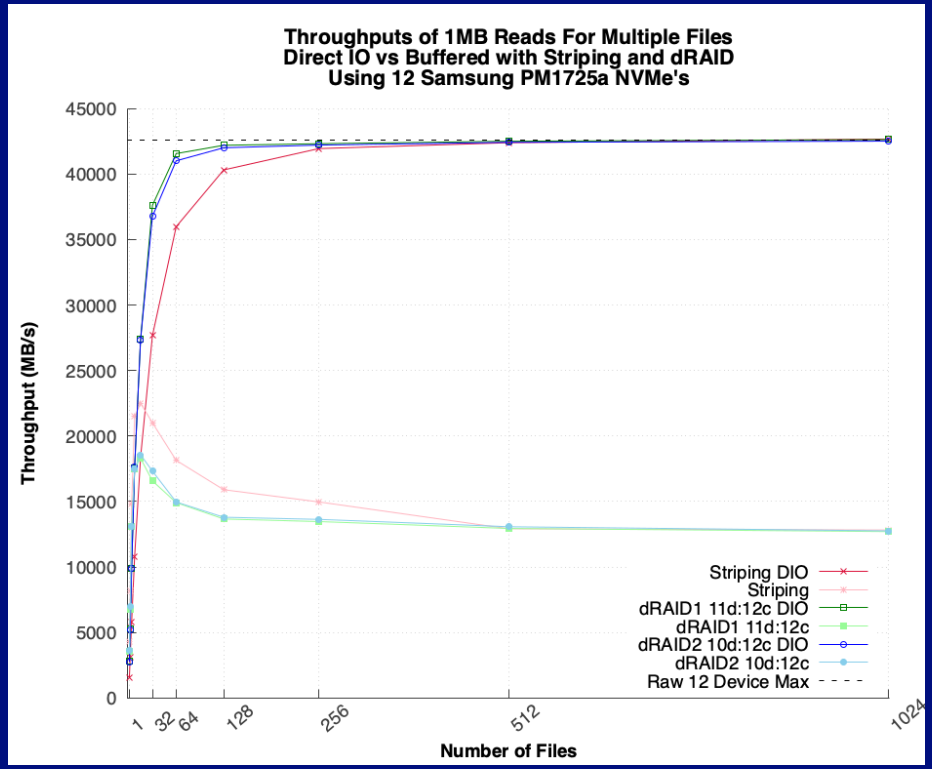
Gen3 Seq. Write Performance Results: Raidz

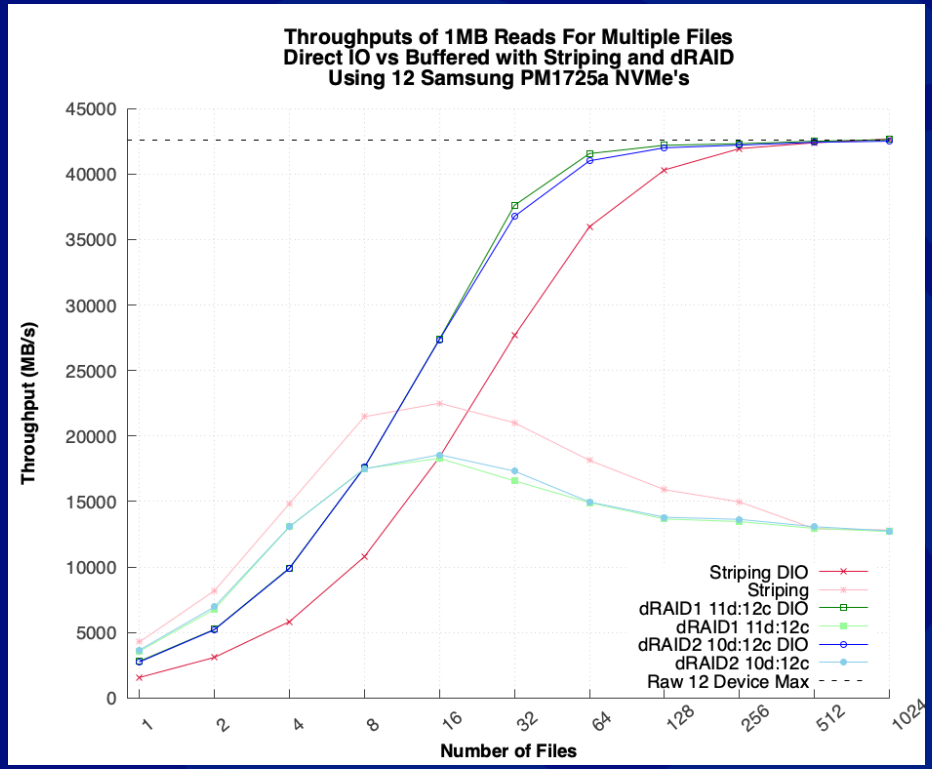




Log Scale

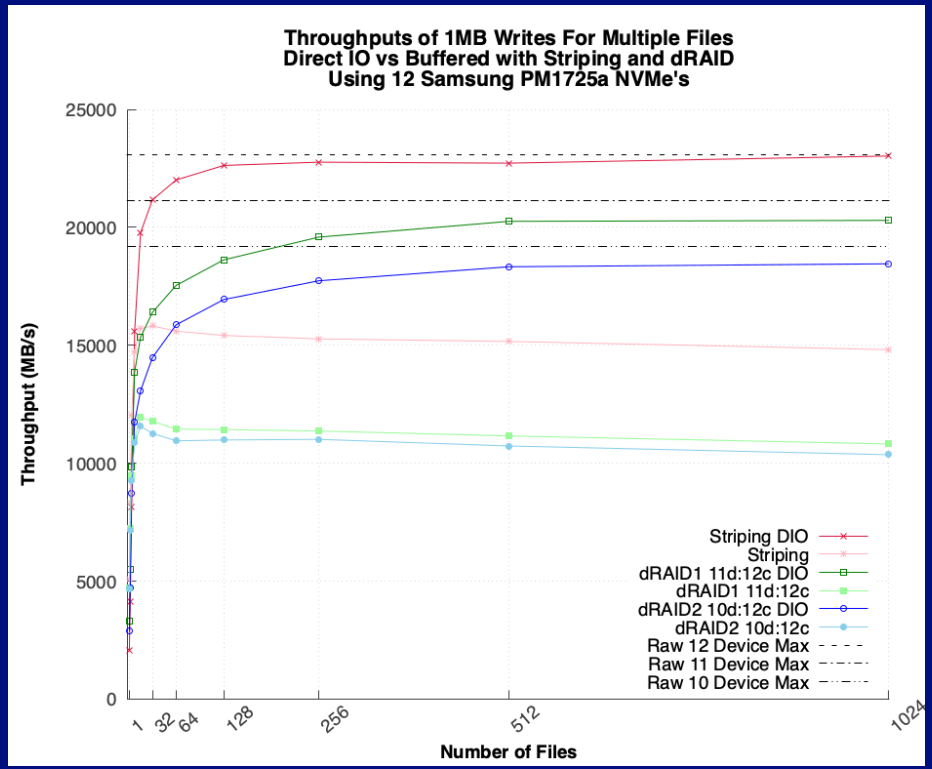
Gen3 Seq. Read Performance Results: dRAID

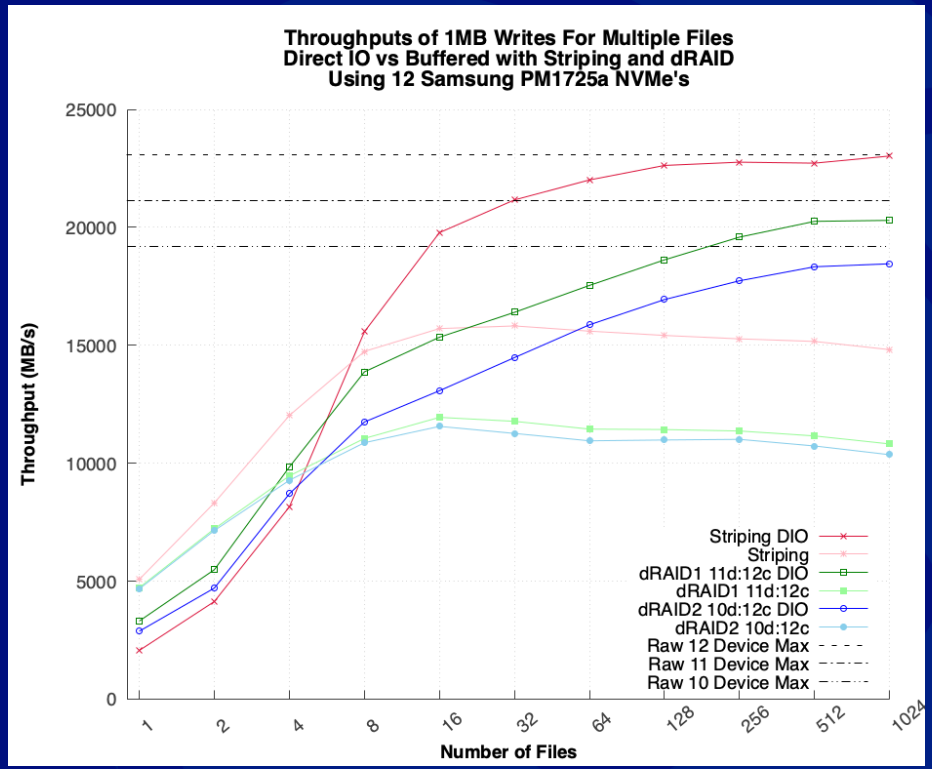




Log Scale

Gen3 Seq. Write Performance Results: dRAID

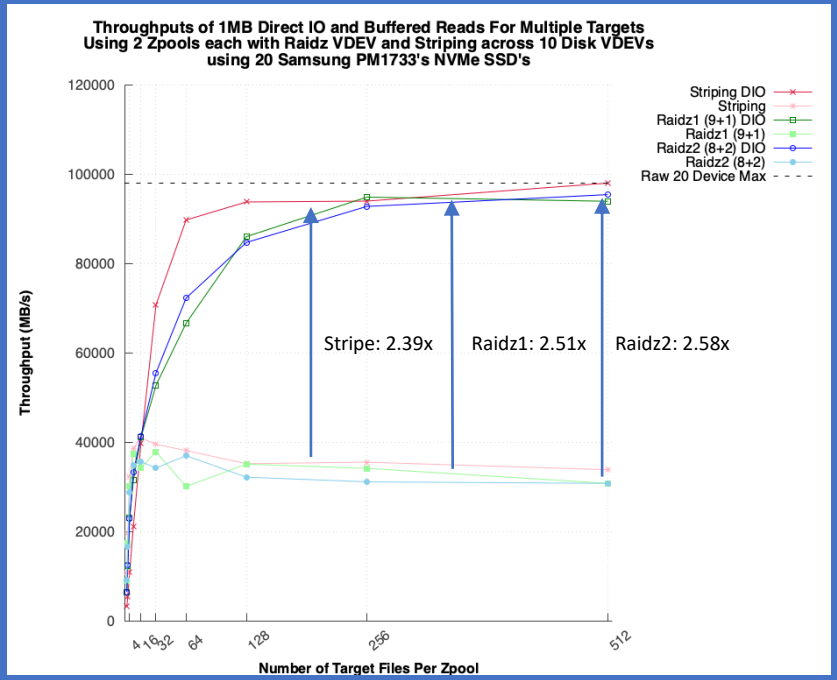




Log Scale

ZFS Seq. Read gen4 Performance Results

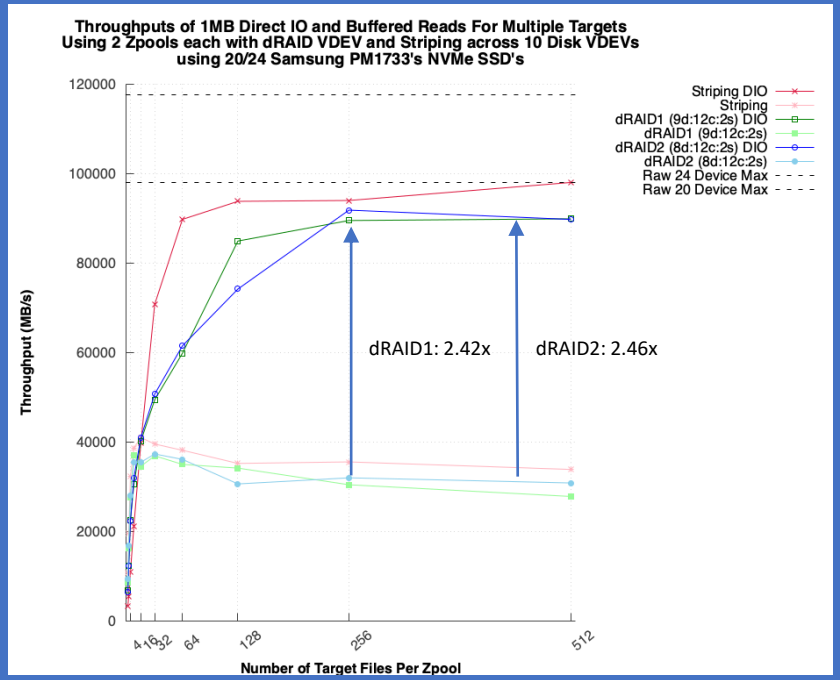
Raidz



Stripe: 99% Raidz1: 97% Raidz2: 97%

Percentage Total Device Bandwidth with O_DIRECT

dRAID

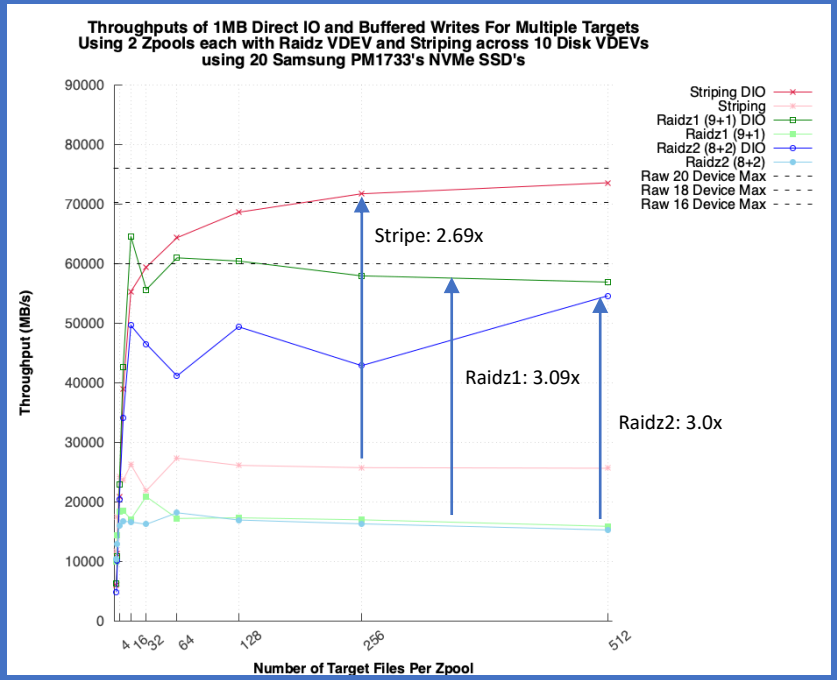


dRAID1: 76% dRAID2: 78%

Percentage Total Device Bandwidth with O_DIRECT

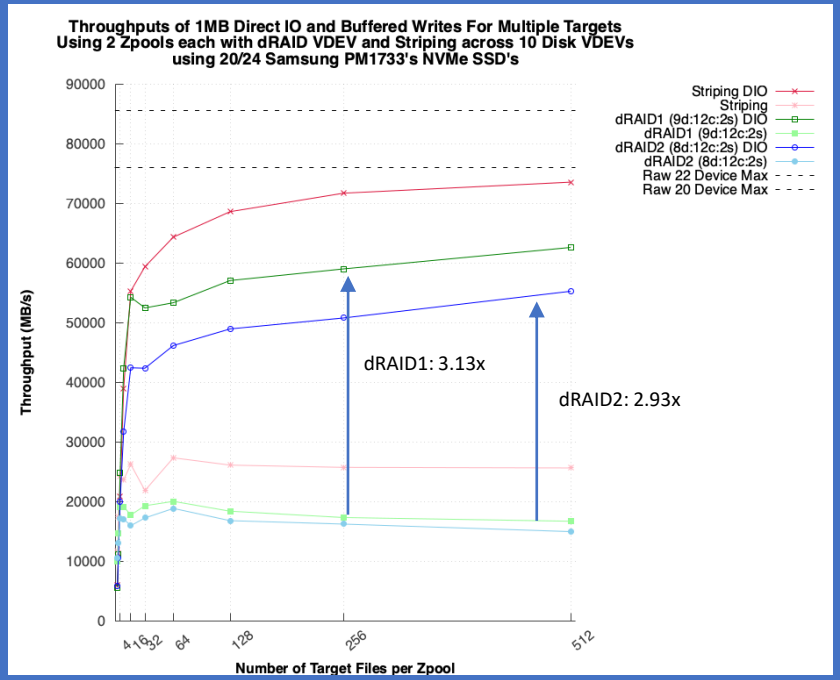
ZFS Seq. Write gen4 Performance Results

Raidz



Stripe: 97% Raidz1: 92% Raidz2: 91%
Percentage Total Device Bandwidth with O_DIRECT

dRAID



dRAID1: 73% dRAID2: 72%
Percentage Total Device Bandwidth with O_DIRECT